99P2795 B4

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 350 149 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 89304046.9

(51) Int. Cl.[4]: H04J 3/06

(22) Date of filing: 24.04.89

(30) Priority: 30.06.88 US 213746

(43) Date of publication of application: 10.01.90 Bulletin 90/02

(84) Designated Contracting States: DE FR GB IT NL

(71) Applicant: DIGITAL EQUIPMENT CORPORATION
146 Main Street
Maynard, MA 01754(US)

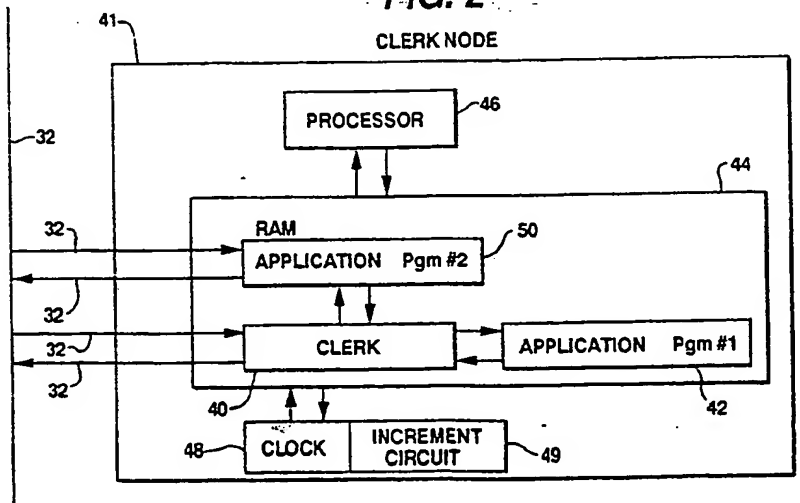
(72) Inventor: Fine, Michael
1 Ballord Place
Cambridge, MA 07139(US)

(74) Representative: Goodman, Christopher et al
Eric Potter & Clarkson 14 Oxford Street
Nottingham NG1 5BP(GB)

(54) Method of maintaining a correct time in a distributed processing system.

(57) A method for maintaining a correct time in a distributed processing system involves clerk nodes maintaining their local clocks by requesting time intervals from server nodes, and server nodes maintaining their local clocks either by requesting time intervals from other server nodes or by receiving time information from an outside source. Server nodes also provide time intervals to requesting clerk nodes and requesting server nodes with such a method. System time is always increasing and monotonic and faulty servers are detected periodically.

FIG. 2



EP 0 350 149 A2

BNSDOCID: <EP___0350149A2_I_>

BACKGROUND OF THE INVENTION

The present invention relates to distributed processing systems, and specifically to a method of maintaining a correct time in a plurality of nodes of a distributed processing system.

a) GENERAL NETWORK OVERVIEW

Some distributed data processing systems are configured as Local Area Networks (LAN's) having multiple nodes connected together for passing information back and forth. One such distributed data processing system network 29 is shown in Fig. 1. The network consists of multiple nodes 30 which pass information along connecting lines 32. The connecting lines 32 may be twisted-pair wires, coaxial cables, or the like. The nodes 30 may be any of various types of devices hooked to the network, such as VAXclusters provided by Digital Equipment Corporation, microcomputers, intelligent workstations, mainframe computers or the like.

All nodes in the network 29 contain software responsible for organizing and coordinating the flow of information between the nodes. One type of such software is dedicated to the mechanics of establishing a signal connection between nodes, of transmitting information between nodes, and of ending the connection when the information transmittal is complete. Such software operates according to one of a variety of communication protocols, such as Digital Network Architecture (DNA) or Open System Interconnection (OSI), which are well known in the art and will not be described herein.

A second type of software in the nodes is dedicated, not to the mechanics of information transmittal, but to generating the information being transmitted, i.e., to application programs. Examples of application programs include word processors, mail programs, data base managers, etc.

A third type of software in the nodes provides an easy-to-use interface between the application programs and various other hardware or software. Software programs of this third type are called "clerks," "clerk software," or "clerk programs" because they handle the details of information transfer for the application programs. Application programs interfacing with clerks are called "clients," "client software," or "client programs" because the application programs are being served by the clerks. Nodes containing clerk software are called clerk nodes.

Fig. 2 shows a clerk program 40 in a memory 44 of a node 41. Clerk program 40 provides an interface between the network via lines 32 and an application program 42, which is also in memory 44 of node 41. Clerk program 40, which is executed by a processor 46 in the node 41, transmits and receives information to and from network 29 via lines 32. Clerk program 40 communicates with application program 42 via locations (not shown) in memory 44 under control of application program 42. Many types of clerk programs similar to clerk program 40 may exist to handle the various needs of application programs.

Fig. 2 also shows a local clock 48 of node 41. Local clock 48 is incremented regularly (if it is digital) or increased at a regular rate (if it is analog) to keep what is called "local time" for node 41. This incrementing is accomplished by an increment circuit 49. Local clock 48 is accessed, i.e., set and read, by clerk program 40 in node 41. Clerk program 40 also serves as an interface to local clock 48 for application program 42. Thus, clerk program 40 of Fig. 2 is called a "time service" clerk because it services requests for the local time from application program 42.

As shown in Fig. 2, not all application programs interface to the network via clerks. Some application programs, such as an application program 50 of Fig. 2, handle their own network communications yet still access clerk program 40 to obtain the local time.

Several methods exist for controlling the communication of information betveen the nodes 30. For example, the network of Fig. 1 uses a method called "carrier-sensing multiple access with collision detection" (CSMA/CD) as defined in IEEE Standard 802.3. Under CSMA/CD, all nodes on the network are of equal importance. A node 30 can "send" information on the network only when no other node is trying to send information. If a node tries to send information at the same time that another node is also trying to send information, each node will wait a randomly-determined period of time and then attempt to resend its information. Since the period of time each node waits before attempting a resend is randomly determined, it is probable that both resends will be successful. Every node on the CSMA/CD network "listens" to all information sent over the network. If any node "sees" information intended for that node, the node will receive the information and take appropriate action. Information passing between the nodes of the network is organized into packets. Each packet contains data or control information surrounded by control and routing information.

VSDOCID: <EP___0350149A2_I_>

Network 29 of Fig. 1 operates using the Ethernet standard for data transmission. This standard governs the protocol used in transmitting data at a low conceptual level. It is compatible with the OSI and DNA communications protocols.

Packets on an Ethernet network may be sent using one of two basic methods: unicasting and multicasting. In unicasting, a packet is sent from a first node to a second node. The destination address field of the packet indicates that the packet is destined for a single node. In multicasting, a single node broadcasts information to multiple nodes and the destination address field of the packet indicates that the packet is destined for a group of nodes.

b) REPRESENTATION OF TIME IN A DISTRIBUTED PROCESSING SYSTEM

i) Notation

An abstract notion of time is defined by some standard which provides a frame of reference by which all values of time are related. One such standard is Universal Coordinated Time (UTC), an international standard maintained by the International Time Bureau (BIH) and the one in general use throughout the world. Political representations of UTC (such as Eastern Standard Time (EST) or Pacific Daylight Time (PDT)) are functionally equivalent to UTC. A clock is simply a device that provides a measure of UTC. The UTC time standard runs at a rate that is almost constant since it is based on ultra-stable atomic clocks. However, some users of time signals need time that is referenced to the rotation of the earth. This time standard is known as UT1 and is inferred from astronomical observations. To keep UTC and UT1 approximately equal, occasional corrections of exactly one second -- called "leap" seconds -- are inserted into the UTC time scale when necessary.

In order to quantify how well a clock is measuring UTC, it is important to define the concept of "correct" time. Let C_i denote a local clock in an ith node of network 29 and let C(i,t) represent the value of C_i at the time t. (Throughout this specification, times denoted by t refer to UTC). In order to quantify how well a clock is measuring UTC, we introduce several properties of clocks. Most important of these is the property of correctness. The following definitions will be used throughout the remainder of the specification:

Inaccuracy:

A perfect clock is one where $C(i,t) = t, \forall t$. However, a real clock always introduces some error. The inaccuracy of C_i at time t, denoted by $\alpha(i,t)$, is defined as the deviation of C_i from UTC. Hence,
$\alpha(i,t) = C(i,t) - t$.

Drift:

The rate of change of inaccuracy is called the drift and for C_i of an analog clock is given by:
$\delta(i,t) = dC(i,t)/dt - 1$.
C_i of a digital clock is given by:
$|C(i,t + \Delta t) - C(i,t) - \Delta t| < \delta\Delta t \delta i dt$
for some interval Δt.

Monotonicity:

A clock is said to be monotonic if its measure of time is never decreasing, i.e., if
$C(i,t_2) \geq C(i,t_1)$ for $t_2 > t_1$.

Resolution:

The resolution of a clock is the maximum time interval that can elapse without there being any change in the value of the clock. The resolution is denoted by p.

BNSDOCID: <EP___0350149A2_I_>

Skew:

For two clocks i and j, the skew at time t is given by
$\sigma_{ij}(t) = |C(i,t) - C(j,t)|$.

Correctness:

For complete time information, it is not sufficient to supply only the time C(i,t). The inaccuracy must be represented as well. Suppose that a clock reports a positive inaccuracy I(i,t) with the value of time C(i,t). For this combination of time and inaccuracy to be correct, we require that $I(i,t) \geq |\alpha(i,t)|$ meaning that $C(i,t) - I(i,t) \leq t \leq C(i,t) + I(i,t)$.
Time at clock i can be represented as an interval $W(i,t) = [C(i,t) - I(i,t), C(i,t) + I(i,t)]$. As shown in Fig. 3, for an arbitrary interval $V = [x,y]$, we define $\lfloor V \rfloor = x$; $\lceil V \rceil = y$, $\|V\| = (x + y)/2$ (the midpoint of the interval); and $|V| y - x$ (the width of the interval).

Using interval notation, the inaccuracy is given by $I(i,t) = |W(i,t)|/2$. A time interval is said to be correct if $\lfloor W(i,t) \rfloor \leq t \leq \lceil W(i,t) \rceil$, i.e., the interval contains UTC. A set of M local clocks is said to be synchronized if the clocks' intervals overlap, i.e., W(1,t)/) W(2,t),.... W(M,t) ≠ 0. If the clocks are correct, they must be synchronized.

Next, some notation for interval arithmetic is defined. The sum of two intervals is defined as [x,y] + [u.v] = [x + u, y + v]. The sum of an interval and a scalar is identical to the sum of two intervals, one of which has width zero. Thus [x, y] + z = [x,y] + [z,z] = [x + z, y + z]. This sum is equivalent to translating the interval along the real line. The notation [x, y] ± z denotes a "stretching" of the interval along the real line. The notation z = [x - z, y + z].

ii) Clocks

Two basic types of clocks exist: clocks that obtain time from an external source, such as radio signals, and clocks that are self-contained, such as crystal clocks. Clocks of the first type measure UTC with a known bound on the inaccuracy which depends on the propagation delay of the radio signal and implementation specific details. If operating correctly, the time and inaccuracy reported by clocks of the first type define an interval that contains the instantaneous value of UTC. Clocks of the second type supply the time in the form of a scalar and there is no inherent notion of inaccuracy. However, clocks of the second type do have bounds on their drifts, which is denoted by δ max. So, given an initial inaccuracy I_i at time T_i, it is possible to compute the inaccuracy at the time that the clock reads T from $I \leq I_i + (T - T_i)\ \delta$ max. Thus, a self-contained clock can represent time as an interval but this interval increases or "degrades" as time progresses.

In previously existing systems, no provision was made for adjusting the time of a local clock except by resetting it with a new value. As a result, the time could jump forward or backward arbitrarily. Such quantum jumps in local time sometimes resulted in events having an ordering by UTC such as shown in Fig. 4a being timestamped to reflect a different ordering such as shown in Fig. 4b.

c) USES OF TIME IN A DISTRIBUTED PROCESSING SYSTEM

Time, in a data processing system, is used in at least three ways. A first use is to time and order events. In many processes performed by data processing systems (for example, event logging), it is desirable to be able to determine an order of occurrence for several events. A second use for time in a data processing system is to determine an interval of time that has elapsed between two events. Examples include system timeouts, Ethernet packet lifetimes, and I/O timeouts. A third use for time in a data processing system is to schedule future execution of events. If an event is to begin execution at a time near a scheduled time, an inaccuracy of the clock must be known and bounded within certain predetermined limits.

Early data processing systems were centralized mainframe computers and usually maintained a time via a single system clock integral to the system. Various operating system and application programs executed on the centralized system and accessed the system clock when they needed a current time value. An error in the time kept by the system clock was discovered by human inspection and remedied simply by

SDOCID: <EP___0350149A2_I_>

manually resetting the system clock whenever the error in its time exceeded some predetermined error limit. For example, the local clock of a VAX data processing system, manufactured by Digital Equipment Corporation, may drift by as much as nine seconds a day. This could cause a drift of as much as 3285 seconds (54.75 hours) per year. Such errors, while inconvenient are easily detected and do not usually cause a major problem for a centralized data processing system. Although such errors cause the time kept by the system clock to deviate from UTC, the time is still increasing and monotonic because the system contains only a single clock. Thus, events will still be ordered correctly and time intervals and timeouts, while perhaps of slightly incorrect duration, will still be correct relative to other intervals and timeouts in the system. Events may be timestamped incorrectly in relation to UTC, but their ordering will not be altered.

In contrast, disagreement between the local clocks in the nodes of a distributed processing system, such as the network of Fig. 1, can cause catastrophic errors in the system. Assume, for example, that both node A and node B of Fig. 1 contain a local clock keeping a local time. A difference in the times kept by the clocks of the two nodes may cause a distributed software process to produce erroneous results.

In Figs. 5a-5f, node A and node B of a distributed processing system are to perform different steps of the following process at UTC times time1 and time2, respectively,

UTC time1:	copy file1 to file2	(by node A)
UTC time2:	copy file3 to file1	(by node B)

where UTC time1 precedes UTC time2. In this algorithm, the order of performance of the steps is critical.

If, as shown in Fig. 5a, the clocks of both nodes A and B are keeping local times that are the same as UTC, the copy from file1 to file2 is performed first (by node A at node A's local time1) and the copy from file3 to file1 is performed second (by node B at node B's local time2). This is the desired order of performance, yielding the results labeled "Results after Step 2" in Fig. 5b.

If, however, the local clock of node B is keeping incorrect time, erroneous results may occur. If, as shown in the timing diagram of Fig. 5c, the local clock of node B is being incremented more frequently than the local clock of node A, then the local clock of node B may reach a local time2 before the local clock of node A reaches a local time1. Similarly, as shown in the timing diagram of Fig. 5d, the local clock of node B may have been mis-set at some previous time. Therefore, even though the local clock of node B is being incremented at correct intervals, local time2 of node B precedes local time1 of node A. Alternately, as shown in the timing diagram of Fig. 5e, the amount by which the local clock of node B is being incremented may be larger than the amount by which the amount by which node A is being incremented. Again, such a condition will cause local time2 of node B to precede local time1 of node A.

If any of the conditions of Figs. 5c, 5d, or 5e exist, the copy from file3 to file1 is performed first (by node B) and the copy from file1 to file2 is performed second (by node A). This inversion of the order of the Steps performed by nodes A and B causes the erroneous results labeled "Results after Step 2" in Fig. 5f.

What is needed is a method for maintaining a correct time in a data processing system where the method ensures that the local clocks of the nodes in the system agree on a correct time. Agreement among the nodes on some random time is not acceptable since the agreed upon time may differ from UTC. A system where all the nodes agree on a non-UTC time will avoid the problems of Figs. 5a-5f, but will cause errors in time-stamping functions, and will not execute scheduled events at the expected times. In addition to maintaining a correct time, such a system should be able to survive the loss or erroneous operation of one or more nodes of the system and to identify and correct these faulty nodes if possible. The times kept by the local clocks of such a system should always be monotonic and increasing to avoid the problems of Figs. 4a-4b.

SUMMARY OF THE INVENTION

The present invention overcomes the problems and disadvantages of the prior art by providing correct time from server nodes to clerk nodes at the request of the clerk nodes.

In accordance with the purpose of the invention, as embodied and broadly described herein, the invention comprises a method for maintaining correct time in a clerk node in a distributed processing system also containing a plurality of server nodes. The clerk and server nodes are connected together and the server nodes provide time information to the system. Each of the nodes includes a local clock keeping a local time and each of the local clocks has a local drift value associated therewith representing a bound on the rate at which that local time deviates from a correct time value. The method comprises the steps

BNSDOCID: <EP___0350149A2_I_>

performed by the clerk node of: requesting an updated time interval from at least one of the server nodes when the local inaccuracy of the local clock of the clerk node exceeds a predetermined maximum inaccuracy value; noting a request time according to the local clock of the clerk node when the clerk node requests the updated time interval; receiving from the server nodes respective updated time interval representations and corresponding delay values; noting, for each receipt of the updated time interval representations and corresponding delay values, a different response time according to the local clock of the clerk node; calculating a correct time interval from the received updated time interval representations, the received delay values, the noted request time, the noted response times, and the resolution value of the local clock of the clerk node, wherein the correct time is contained within said calculated correct time interval; and adjusting the local time kept by the local clock of the clerk node according to the calculated correct time interval.

The invention further comprises a method for maintaining a correct time in a clerk node including the steps performed by the server node of: receiving a request for an updated time interval from another node when the local inaccuracy of the local clock of the other node exceeds a predetermined maximum inaccuracy value; noting a receipt time according to the local clock of the server node when the server node receives the request from the other node; calculating a delay value from the noted receipt time; calculating an updated time interval representation from the local time of the local clock of the server node and the inaccuracy of the local clock of the server node; and sending the updated time interval representation and the corresponding delay value to the other node.

The accompanying drawings, which are incorporated in and constitute a part of the specification, illustrate one embodiment of the invention and, together with the description, serve to explain the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a general block diagram of a local area network.

Fig. 2 is a block diagram of a clerk node which can be connected to a network.

Fig. 3 is a diagram used to explain terms which define a time interval.

Figs. 4a and 4b are diagrams used to illustrate a problem which may be caused by certain methods of resetting clocks in nodes of a network.

Figs. 5a-5f are diagrams used to illustrate a problem which may be caused by timing malfunctions in nodes of a network.

Fig. 6 is a block diagram of a time maintainer node of a network in accordance with this invention.

Fig. 7 is a block diagram of a time provider node of a network in accordance with this invention.

Fig. 8 is a flow chart of one method in accordance with this invention of adjusting a local clock in a clerk node of a network.

Fig. 9 is a flow chart of further steps of the method of Fig. 8 in accordance with this invention of adjusting a local clock in a clerk node of a network.

Fig. 10 is a diagram illustrating the time delays in a request from a clerk and a response from a server.

Fig. 11 is a flow chart of the preferred procedure by a clerk for requesting a time update from a server.

Fig. 12 is a flow chart of part of a preferred method of the present invention to compute the smallest interval guaranteed to contain UTC.

Fig. 13 is a flow chart of a preferred method of efficiently finding the minimum endpoint of the best interval.

Fig. 14 is a flow chart of a preferred method of the present invention for adjusting for leap seconds.

Fig. 15 is a flow chart of a preferred method of the present invention of how a server node responds to a request for time from a clerk node.

Fig. 16 is a flow chart of a preferred method of the present invention for manually resetting time maintainer nodes.

Fig. 17 is a flow chart of a preferred method of the present invention for detecting faulty server nodes.

DESCRIPTION OF THE PREFERRED EMBODIMENT

NSDOCID: <EP___0350149A2_I_>

Reference will now be made in detail to preferred embodiments of this invention, examples of which are illustrated in the accompanying drawings. Whenever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

a) HARDWARE OVERVIEW

The method of the present invention is preferably carried out by data processing hardware executing a plurality of software programs. That hardware is preferably contained in nodes of a network such as network 29 shown in Fig. 1. As shown in Fig. 2, a node 41 which executes clerk program 40 is called a clerk node. The purpose of clerk node 41 is to service requests for a local time of local clock 48. As explained above, information passes from the connecting lines 32 into the clerk node 41 and to clerk program 40 executing on clerk node 41. Clerk program 40 updates its local clock 48 based on information received over lines 32 and, in response to requests from application program 42, reads the local time from local clock 48 and supplies it to application program 42.

Preferably, other nodes of the network execute similar software programs to clerk program 40, and supply time information to other application programs.

In accordance with the present invention, other nodes, called server nodes, execute software programs called server programs. Server programs send time information to clerk and server nodes over lines 32. Preferably, there are two types of server nodes: time maintainer nodes (also called time maintainers) and time provider nodes (also called time providers).

Fig. 6 shows an embodiment of a time maintainer node 80. Time maintainer node 80 is connected to the network via lines 32. Similar to clerk node 40, time maintainer node 80 contains a processor 82 and a local clock 84. Local clock 84 contains an incrementation circuit 88 for periodically increasing the local time of local clock 84 by a predetermined resolution value.

Upon receiving a request for an updated time from a clerk node over the network, processor 82 executes a server program 86, which is stored in a memory 87 (preferably RAM) of time maintainer node 80. Server program 86 sends time information from server node 80's local clock 84 a to requesting clerk node over the network.

Some time maintainer nodes may also include a battery back-up clock and non-volatile memory. The non-volatile memory stores a base local time and inaccuracy at a time T_r, which can be recovered in case of system failure. Similarly, the battery back-up clock continues keeping a local time during system failure.

Fig. 7 shows a second type of server node 80′ called a time provider node. Like time maintainer node 80 of Fig. 6, time provider node 80′ contains a processor 82′ and a local clock 84′. A server program 86′, which is stored in a memory 87′ of the time provider node 80′ and executed by processor 82′ of the node 80′, sends time information from the local clock 84′ over the network upon receiving a request from a clerk node.

Preferably a requesting clerk node does not perceive any difference between time maintainer node 80 and time provider node 80′. Clerk programs use identical protocols to communicate with both types of nodes and the information received by a clerk program from both types of server nodes preferably has the same format.

The key difference between a time maintainer node, such as node 80, and a time provider node, such as node 80′, lies in the way those nodes update their local clocks. Time maintainer node 80 updates its local clock using the same method as clerk nodes, which is described in detail below. Time provider 80′ updates its local clock 84′ by accessing an external time source.

As shown in Fig. 7, time provider node 80′ contains a radio transmitter/receiver 94 which is used to receive time signals from a UTC broadcasting source, such as the WWV in Denver, Colorado. These received time signals are used by server program 86′ to update local clock 84′ of time provider node 80′. It should be noted that time provider nodes could alternatively access some other independent time source outside the network that provided time information not contaminated by any system delays, drifts of the local clock, or other inaccuracies. Preferably, a time provider node that cannot access its external source will alternately update its local clock using the same method as time maintainer and clerk nodes.

b) OPERATION OF CLERK NODES

A method according to this invention will be described with respect to clerk node 40 shown in Fig. 2, although it should be understood that a network using the present invention may include other clerk nodes

BNSDOCID: <EP___0350149A2_I_>

operating according to the method described below. The description of the operation of clerk nodes in accordance with the present invention will be presented in two parts. The first part describes how a clerk maintains its local clock and knowledge of the inaccuracy of its clock. This first part includes the method by which a local clock of a clerk node is correctly and monotonically adjusted to compensate for drift. The second part describes how a clerk node resynchronizes (obtains a new estimate of the time from the server nodes). This second part describes under what circumstances a clerk resynchronizes, the protocol for obtaining time estimates from server nodes, and an algorithm for computing an optimal correct time interval from these estimates.

i) Monotonically Adjusting the Local Clock in Clerk Nodes

A local clock, such as local clock 48 of Fig. 2, preferably comprises a software register representing the current time. This register preferably is updated by incrementation circuit 49, which may comprise a service routine invoked by a hardware interrupt of a periodic timer. In such an embodiment, these interrupts are called "ticks." The time between ticks determines the resolution p of the local clock, which, in a preferred embodiment, equals 10 ms. An interrupt service routine services tick interrupts and increments the local clock by p. By changing the amount of the increment while keeping the time between ticks constant, the clock drift can be artificially altered, causing local clock 48 to gain or lose time. Alternatively, one could alter the time between ticks while keeping the software increment at a predetermined value.

The preferred manner by which clerk node 40 adjusts its local clock 48 to a new time can be described by the procedures of Figs. 8 and 9. Fig. 8 shows a flow chart for a procedure called "AdjustClk_Start," which is executed at a "resynchronization time," T_r, to begin incrementing the local clock by a new incrementation value instead of by the predetermined value p. Fig. 9 shows a flow chart for a procedure called "AdjustClk_End," which is executed at a time $T_r + D$ (where D is the duration for which the new incrementation value is used) to end incrementing the local clock by the new incrementation value and resume incrementing by p.

In Figs. 8 and 9, clerk program 40 has determined that the actual time will be $<T_a,I_a>$ when local clock 48 reads T_r. "Maxdrift" represents the predetermined minimum number of seconds during which the local clock will normally drift one second. The notation $<T_a,I_a>$ represents a time interval $[Ta - I_a, T_a + I_a]$ having time t_a as a midpoint and extending an inaccuracy distance I_a in both a positive and negative direction. Thus, maxdrift = $1/\delta$max. The value "k" represents a predetermined drift increment rate, i.e., the rate at which the drift of the local clock will be artificially increased or decreased during the interval [Tr,Tr + D]. In the preferred embodiment, $k \geq -1$. In other words, a local clock cannot be decreased at a rate of more than one second per second or it will run backwards. In Figs. 8 and 9, "adj" represents a predetermined number of seconds during which the drift of the clock is altered by one second. (Thus, adj = $1/|k|$ and, in the preferred embodiment, adj $\geq$ 1.) "I_{rate}" represents a rate at which the inaccuracy of the local clock increases in response to the drift of the local clock. Under normal operation, $I_{rate} = \delta$max, i.e., I_{rate} equals the maximum drift of the local clock. However, while the local clock is being adjusted I_{rate} is decreased to compensate for the drift increment rate, k.

First, as shown in Fig. 8, at the resynchronization time T_r, a current time, T_i, is set to T_r (Step 100). An inaccuracy, I_i, corresponding to T_i, is set to:

$I_i = I_a + |Ta - Ti|$

(Step 100), where I_a is the inaccuracy of the new time value and $|T_a - T_i|$ is an additional inaccuracy representing the difference between the local clock and the actual time.

If T_r is equal to T_a (Step 104), then the local clock is already set correctly and no further adjustment is needed. If T_r is not equal to T_a, however, and if T_a is greater than T_r (Step 106), then k, the rate at which to alter the drift of the local clock is positive (Step 108) so that the local clock will gain time. Otherwise, k is negative (Step 110) so that the local clock will lose time.

Next, I_{rate}, the rate at which the inaccuracy of the local clock increases in response to the drift of the local clock, must be set to reflect the changed drift value of the local clock. I_{rate} is calculated according to the following formula:

$$I_{rate} = \frac{adj - maxdrift}{adj * maxdrift}$$

ISDOCID: <EP___0350149A2_I_>

(Step 112). Because, in the preferred embodiment, the absolute value of the adjustment rate, $|k|$, is greater than the drift rate, δmax, adj < maxdrift, i.e., $1/k < 1/\delta max$, and therefore I_{rate} is negative and I_i, the inaccuracy before adjustment, is less than I_i', the inaccuracy after adjustment.

"D," the duration of the adjustment is computed in Step 114 using the formula:

$D = |(T_a - T_i)| * adj.$

Because adj equals the predetermined number of seconds during which the drift of the local clock is altered by one second and because $|T_a - T_i|$ equals the total number of seconds by which the clock is to be altered, the product of adj and $|T_a - T_i|$ yields D. The procedure AdjustClk_End is then scheduled to execute at time $T_r + D$ (Step 116) and clock increment circuit 49 begins incrementing local clock 48 by $(1 + k)p$ (Step 118).

It is necessary that the procedure AdjustClk_Start completes execution before the next tick after T_r. Otherwise, the procedure would have to compensate for its own execution time.

For the purpose of example, suppose that the variables p, k, δmax, adj, maxdrift, T_a, I_a, and T_r have the following values:

$p = 1.2$ sec
$k = 4/5$ sec per second
$\delta max = 1.4$ sec per second
$adj = 1/k = 5/4$ sec
$maxdrift = 1\ \delta max = 4$ sec
$T_a = 5$
$I_a = 3$
$T_r = 1$
Then, $I_i = I_a + |T_a - T_r|$
$= 3 + |5 - 1| = 7$ sec

$$I_{rate} = \frac{adj - maxdrift}{adj * maxdrift}$$

$$= \frac{5/4 - 4}{5/4 * 4} = -11/20 \text{ sec per second}$$

$D = |(T_a - T_i)| * adj$
$= |(5-1)| * 5/4 = 5$ sec.

In other words, for local clock 48 to gain 4 seconds, its drift must be increased by 4/5 seconds for a total duration of 5 seconds. Thus after 5 seconds (n = 10 increments),

$T_i' = T_r + n(1 + k)p$
$= 1 + 10(1 + 4/5)1/2 = 10.$

Local clock 48 has gained 5 seconds due to normal incrementation and 4 seconds due to its altered drift rate.

The Steps of Fig. 9 return local clock 48 to its regular incrementation procedure. At time $T_r + D$, a new inaccuracy, I_i' is computed according to the formula:

$I_i' = I_i + (T_i' - T_i) * I_{rate}$

(Step 122), where I_i is the inaccuracy from Step 100 of Fig. 8, T_i is the current value of local clock 48, T_i' is the time from Step 100 of Fig. 8, and I_{rate} is the increase in the inaccuracy from Step 112 of Fig. 8. In the current embodiment, because I_{rate} is negative, $I_i' < I_i$. Both T_i' and I_i' are stored in memory 44 to be used by clerk 40 in determining a time to resynchronize as described below.

Therefore, in the above example, the new inaccuracy of local clock 48 is:

$I_i' = I_i + (T_i' - T_i) * I_{rate}$
$= 7 + (10 - 1) * -.55 = 7 - 4.95 = 2.05.$

In Step 128, the amount to alter the clock's drift, k, is set to zero because the clock adjusting procedure is done. Similarly, the rate of increase of the inaccuracy is reset to its original value of δmax in Step 130.

The procedures illustrated in Figs. 8 and 9 thus guarantee that local clock 48 remains monotonic. However, this may not always be desired behavior. In particular, if local clock 48 differs from the correct time by a large value, it may be preferable to simply reset it to the correct time rather than to adjust its drift using the procedures of Figs. 8 and 9.

ii) Determining When Clerk Nodes Should Obtain Time Estimates

BNSDOCID: <EP 0350149A2_I_>

Clerk program 40 resynchronizes its local clock 48 when its inaccuracy exceeds a predetermined value I_{max}. If a clerk program has resynchronized to a correct time T_a with inaccuracy I_a, the inaccuracy at any later time T is given by $I = I_a + (T - T_a)I_{rate}$, where $I_{rate} = \delta max$. However, the preferred embodiment does not guarantee any bound on I_{max}. As a result, it is possible that $I_{max} - I_a$ is small or even negative. To prevent clerk program 40 from synchronizing continuously, the preferred embodiment computes the next time to resynchronize, T, according to the following rule (where "max" is a function that selects a maximum of two values and "Synchold" is a predetermined value representing the shortest time allowed between resynchronizations in the preferred embodiment).

$T = T_a + max(synchold, (I_{max} - I_a)/\delta max$.

$(I_{max} - I_a)/\delta max$ represents a time when the inaccuracy of local clock 48 will equal or exceed I_{max}.

iii) Obtaining Time Estimates from Server Nodes by Clerk Nodes

In the preferred embodiment of this invention, clerk program 40 obtains the time from a particular server node by making an explicit request to the server program of that node for the time and then waiting for a response. This method of access has two features. First, it makes it possible for a clerk to measure the delay incurred due to the communication protocol overhead and, hence, makes it possible to bound the inaccuracy of the received time. Second, it requires no specific underlying communications hardware.

Fig. 10 is a time-space diagram of a request from a clerk, such as clerk program 40, and a response from a server program, such as server program 86 of time maintainer node 80 or server program 86' of time provider node 80'. The times t_i, i = 1,2...8, correspond to the values of UTC (which can never be known exactly due to the delays discussed above.) The value of the clerk's clock at time t_i is denoted by $C_c(t_i)$ and the value of the server's clock at time t_i is denoted by $C_s(t_i)$. A double arrow denotes a message transmission and a single arrow denotes a message reception.

Fig. 11 is a flow chart of a preferred method performed by the clerk node for making the request illustrated by Fig. 10. Once the clerk decides that it is time to resynchronize (Step 200), at t_1, the clerk reads its local clock which returns the value $C_c(t_1)$ (Step 202). The clerk then initiates a request to the server (Step 204) the request is transmitted at time t_2 after an unknown sending delay S_c, as shown in Fig. 10, and the request is received by the server at time t_4 after unknown delays of $\overline{J}_{cs}$ and r_s.

The clerk waits until it receives a response from the server (Step 208) in the form of an interval [u,v], and a delay w. (In a preferred embodiment, the interval [u,v] is represented by a midpoint value and an inaccuracy value. Other embodiments could represent [u,v] by two endpoints.) The response from the server is received at time $C_c(t_8)$ (Step 210) after incurring random delays similar to those of the request, namely S_s, $\overline{J}_{sc}$, and r_c.

Ideally, the clerk would compute the time $C_s(t_1)$ from the value $C_s(t_4)$ returned with the response as follows:

$C_s(t_1) = C_s(t_4) - x$,

where $x = s_c + \overline{J}_{cs} + r_s$. However, x is the part of the delay that cannot be known exactly. It is in the range $0 \leq x \leq t_8 - t_1 - w$. If $x = 0$, then $C_s(t_1) = C_s(t_4)$ while if $x = t_8 - t_1 - w$, then $C_s(t_1) = C_s(t_4) + w - (C_c(t_8) - C_c(t_1))$. Hence, at best, the time seen by the clerk at t_1 can be bounded by

$C_s(t_4) + w - (C_c(t_8) - C_c(t_1)) \leq C_s(t_1) \leq C_s(t_4)$,

but it can never be known exactly. The effect of the drift of the clerk with respect to the server over the time period of the exchange is neglected. Clearly, by minimizing the components of delay, or by compensating for known delays such as w, the interval bounding $C_s(t_1)$ can be reduced.

Since local clock 48 is discrete, the timestamp $C_c(t_8)$ actually corresponds to the value of local clock 48 in the range $[C_c(t_8), C_c(t_8) + p]$, where p is the resolution of local clock 48. Furthermore, the time provided by server i is in fact an interval $W(i,t_4) = [u,v]$ rather than a scalar value. Thus, the time interval of S_i at time t_1 as seen by the clerk is given by:

$U(i,t_1) = [u + w - (C_c(t_8) + p - C_c(t_1)), v]$

In the preferred embodiment of the invention, the clerk complies $U(i,t_i)$ in this manner (Step 212).

The value w, which is returned with the time to the clerk, corresponds to the known delay at the server after the time which is reported to the clerk (and hence, it may include any deterministic component of S_s). The server does not include any deterministic component of r_s in the value of w returned as this delay occurs before t_4. Furthermore, there is no need to return the value of known delays before t_4 since the server can include this information by decrementing the time returned (i.e., $C_s(t_4)$) and by incrementing w by the same amount. It would not be incorrect to report a value of w less than the actual delay incurred. This smaller value of w would merely cause a larger than necessary inaccuracy in local clock 48. It would,

ISDOCID: <EP___0350149A2_I_>

however, be incorrect to report a value of w greater than the actual delay. This larger value of w would cause the inaccuracy of local clock 48 to be smaller than could be justified by the actual delay incurred by the server. To maximize performance, w should reflect the delay at the server as accurately as possible, but w should never be greater than the actual delay incurred.

Since request and response packets may be lost or servers may become unavailable, a clerk waits only a predetermined time for a response to a request.

iv) Computing the Correct Time from Received Time Estimates in Clerk Nodes

Fig. 12 is a flow chart of the preferred method used by clerk program 40 in accordance with of the present invention to compute the smallest interval guaranteed to contain UTC. Assume that clerk C_i obtains the time from M servers by using the method in Fig. 11 and, therefore, has a set of intervals $U(j,T_j)$ $j = 1,2...M$. T_j is the value of the clerk's clock at some arbitrary instant t_j (i.e., $T_j = C(i,t_j)$) and $U(j,T_j)$ is the value of the j^{th} server's clock at t_j, as seen by the clerk.

First, procedure AdjustClk-End of Fig. 9 is invoked (Step 250). This ensures that the clock is drifting at a rate less than maxdrift (i.e., $k = 0$). Next, for each interval received from a server, its value at the next resynchronization time T_r is computed from

$U(j,T_r) = U(j,T_j) + (T_r - T_j) \pm (T_r - T_j)\delta max$

(Step 252).

Next, all the endpoints of the intervals $U(j,T_r)$ are arranged into a list in ascending order (Step 254). The list is of length 2M because there are two endpoints for each interval in the list. If two or more endpoints have the same value, those intervals corresponding to lower bounds must preceed those intervals corresponding to upper bounds in the list. That is, if $\lfloor U(i,T_r) \rfloor = \lceil U(j,T_r) \rceil$, then $\lfloor U(i,T_r) \rfloor$ must preceed $\lceil U(j,T_r) \rceil$ in the list.

Next, a counter, f, representing the number of existing faulty servers is set to zero (Step 256) and the list is scanned in ascending order to find the first endpoint that is contained in at least M - f intervals (Step 258). This point, if found, is a lower endpoint of the smallest time interval containing UTC. If no such point is found then there are more than f faulty servers. The counter f is incremented (Step 260), control returns to Step 258, and a smaller number of intervals is scanned for a common endpoint. If the lower endpoint is found (Step 258), then the list is scanned in descending order to find a second endpoint that is contained in at least M - f intervals (Step 262). This point is an upper endpoint of the smallest time interval containing UTC.

Scanning the list for the first upper and lower endpoints is accomplished more easily if the elements of the list are marked as maximum end points or minimum end points depending on whether they correspond to $\lceil U(j,T_r) \rceil$ or $\lfloor U(j,T_r) \rfloor$, respectively.

Fig. 13 contains a flow chart describing a method for finding the lower endpoint of the best correct time interval efficiently. This method is used in Step 262 of Fig. 12. First, two counters, IntersectingInts and MostInts, are initialized to "1." IntersectingInts represents a number of intervals containing a tentative minimum endpoint. MostInts represents number of intervals containing a current minimum endpoint. A third loop counter, i, is initialized to "2" to indicate that the second list element is being considered as the tentative minimum endpoint. The first element in the ordered list of endpoints of Step 254 of Fig. 12 is designated as the current minimum endpoint by setting Minpoint to "1" (Step 270). Thus, the initial value of IntersectingInts indicates that only one interval thus far contains the tentative minimum endpoint indicated by the counter i. The initial value of MostInts indicates that the first endpoint in the list (designated by Minpoint) is contained in one interval. Next, for increasing values of i through 2M-1, the following Steps are performed.

If the ith list element is greater than the tentative minimum endpoint designated by Minpoint (Step 272), then the counter IntersectingInts is incremented (Step 274), and a test is performed to determine if the counter IntersectingInts is greater than the counter MostInts (Step 276). If yes, the counter MostInts is set to IntersectingInts, the number of intervals containing the ith list element is saved, and the current value of i is saved as the tentative minimum endpoint (Step 278). Otherwise, if IntersectingInts is not greater than MostInts, control passes to Step 282.

If the ith list element is not greater than the tentative minimum endpoint (Step 272), then the counter IntersectingInts is decremented (Step 280) and control passes to Step 282.

At Step 282, a test is performed to determine if more endpoints in the list have yet to be examined. If yes, i is incremented (Step 284) and Steps 272 through 282 are repeated to examine another endpoint. If no, then all of the 2M-1 elements in the list of endpoints have been checked and control passes to Step

BNSDOCID: <EP___0350149A2_I_>

286. At Step 286, a test is performed to determine if an endpoint within M-f intervals has been found. If yes, a minimum endpoint has been found (Step 288). Otherwise, no endpoint is contained in at least M-f intervals (Step 290). A similar algorithm can be used to find the maximum endpoint of the best correct time, as required by Step 262 of Fig. 12, by searching the endpoint list in descending order.

The result of the method of Figs. 12 and 13 is the time (with inaccuracy) that the clerk's clock should read when the clock value becomes T_r, i.e., $<T_a,I_a>$. A clerk preferably uses the times from all the servers that are available. Other embodiments, however, may use the times from fewer than all of the servers.

v) Leap Seconds In Clerk Nodes

All clerks become faulty immediately when leap seconds occur unless their inaccuracies are sufficiently large so that their intervals contain UTC both before and after the leap second. To ensure correctness in the presence of leap seconds, clerks must increase their inaccuracies by one second whenever a leap second might occur. Unfortunately, leap seconds may occur at any point in time, the only requirement being that at least eight weeks warning be given by the International Time Bureau (BIH) of an impending leap second. However, when choosing the time of a leap second, BIH is required to give preference to the last second in the months of June and December. Secondary preference is given to the last days of March and September. To simplify the matter, the preferred embodiment assumes that leap seconds occur at all four possible times.

Fig. 14 is a flow chart for making adjustments for leap seconds by clerk program 40. If a current time, T, of local clock 48 plus a current inaccuracy, I, of local clock 48 is midnight of either March 31, or June 30, September 30, or December 31 (Step 300) then the current inaccuracy, I, is incremented by "1" (Step 304). This adjustment is always performed whether or not a leap second actually occurs. Other embodiments may compensate for leap seconds only in response to a command from a human operator.

c) OPERATION OF SERVER NODES

A description of the operation of server nodes consists of four parts. These parts are: how servers initialize and operate; how time maintainer nodes periodically obtain a new time from other servers; how time maintainer nodes can be manually reset; and how servers periodically check for faulty servers.

i) Response to Clerk Node's Requests by Server Node

Fig. 15 is a flow chart performed by a server in responding to a clerk node's requests, as demonstrated in Fig. 10. As shown in Fig. 15, a server waits until it receives a request from a clerk (Step 220) and notes a time of receipt $C_s(t_4)$ (Step 222). This time of receipt is later than the actual arrival time of the request by r_s. The server processes the request by computing its own inaccuracy at time $C_s(t_4)$ and by noting a known component of delay w (Step 226). Lastly, the server transmits the local time interval and w to the requesting clerk at time $C_s(t_5)$ (Step 228). This action takes a time S_s. As explained above, delays r_s and S_s are random and unpredictable.

ii) Initialization of Server Nodes

A time provider initializes by obtaining the time from an external source. It then attempts to discover other servers, by means of the protocols described below, for the purpose of periodically checking for faulty servers.

When a time maintainer initializes, it first attempts to discover other servers, as described below. If other servers are available, it obtains the time from them as if it were a clerk node. If no other servers are available (meaning that the initializing server is the first server to be started), but the time maintainer has a battery back-up clock and non-volatile memory in which T_l and I_l are stored (as shown in Fig. 6), the time maintainer sets its local clock to the time of the battery clock and restores T_l and I_l from the stored values.

If the timer maintainer does not have a battery clock, it waits either for other server nodes (such as time providers) to become available or for a command from a human operator to reset the time. In the latter case, the time maintainer simply accepts the new time without recourse to any consistency checks.

ISDOCID: <EP___0350149A2_I_>

iii) The Synchronization Algorithm for Server Nodes

Time maintainers resynchronize periodically to compensate for the drift of their local clocks and to reduce the width of their time intervals. A time maintainer determines when to synchronize by the same method used by clerks, which is described in connection with Figs. 10-13. When resynchronizing, a time maintainer also performs the algorithm described below for detecting faulty servers and, in addition, adjusts its clock to the correct time in the same way that clerks do (see Figs. 8-9). In contrast, time providers do not resynchronize to other servers because time providers obtain time from an external source. The inaccuracy of the time obtained by time providers is implementation specific because it depends on the hardware used to obtain the external time.

iv)Manual Resetting of Time Maintainer Nodes

If there are no time providers on the network, there is nothing to limit the extent to which the inaccuracies of the time maintainers increase. Thus, after long periods of time (for example, on the order of weeks), the inaccuracy of a time in a network may be large (on the order of minutes). To prevent the inaccuracy from becoming undesirably large, a manual reset method exists in the preferred embodiment whereby the time can be updated by resetting the time at single time maintainer. To protect against a human operator erroneously supplying an incorrect value of time, the manual reset method requires the time interval supplied by the human operator to be contained in the local time interval of the server being reset. Thus, the manual reset method described below does not allow large changes to the current time kept by a local clock. It does, however, allow a human operator to reduce the inaccuracy of a local clock or to make small changes to the value of the clock.

Fig. 16 shows a flow chart of the algorithm that a preferred embodiment of a time maintainer node executes in response to the receipt of manual reset request including a new time interval. The new time interval is denoted by $<T_m, I_m>$ and the time at which the manual reset request is received, as measured by the local clock, is denoted by T_u.

First, the correct time is computed as described in connection with Figs. 12 and 13 (Step 320). This Step determines the actual time $<T_a, I_a>$ corresponding to the future instant at which the local clock will read T_r. The interval $<T_m, I_m>$, which corresponds to local time T_u, is translated to its equivalent value $<T'_m, I'_m>$, corresponding to local time T_r, according to the formula given above in connection with Step 252 of Fig. 12 (Step 326).

If $<T'_m, I'_m>$ is not contained in the interval $<T_a, I_a>$ (Step 328), the manual reset request is rejected and an error is noted (Step 330). Otherwise the request is accepted and control passes to Step 332 where T_a is set to T'_m and I_a is set to I'_m. When the local clock reads T_r, procedure AdjustClk-Start shown in Fig. 8 is initiated to adjust the local clock of the time maintainer node (Step 334).

The time maintainer must complete the adjustment accomplished by the Steps shown in Fig. 16 before resynchronizing to other servers. Otherwise, the manual reset request may not take effect. It is only necessary to reset a single time maintainer. All other time maintainers will conform to the new time once they resynchronize.

When resetting the time, a human operator is expected to supply a correct inaccuracy. If no inaccuracy is supplied, a predetermined default value is used. Human operators resetting the time must supply a correct inaccuracy to within a predetermined inaccuracy to ensure correct operation of the system.

v) Epochs and Epoch Numbers in Server Nodes

Occasionally, the time kept by the nodes in a network is incorrect due to operator error or some catastrophic failure and not to failures in the servers themselves. When all nodes are keeping an incorrect time, it is impossible to reset the time using the manual reset request described above. Instead, it is necessary to shut down all the servers and restart them with a new, correct time interval. This process, however, may not be possible if a faulty remote server fails to shut down. To recover from this condition, the present invention employs epoch numbers which allow groups of servers to be decoupled, and which make it possible to introduce a new value of time into the distributed system by introducing it into one of the decoupled groups.

In the preferred embodiment, each server is assigned an epoch number. Servers preferably include their epoch numbers in their responses to requests for the time. Clerks always ignore this parameter.

EPNSDOCID: <EP___0350149A2_I_>

However, when a server resynchronizes or checks for faulty servers, it ignores servers with epoch numbers different from its own, as shown in Step 402 of Fig. 17, thereby decoupling that server from servers of a different epoch. A server epoch number can only be changed by a human operator. Whenever a new epoch number is assigned to a given server, that server must reinitialize according to the procedure described above. A human operator changes the epoch number at a single server and supplies it with the correct time if the server is a time maintainer node. Then, the human operator can migrate the other servers to the new (correct) epoch by changing their epoch numbers, one at a time, to correspond to that of the new epoch. As each server's epoch number is changed, it will reinitialize and obtain the correct time. To assist with the management of epoch numbers, each server notes any servers whose epoch numbers are different from its own.

vi) Detecting Faulty Server Nodes

Preferably, server nodes periodically check to see if other server nodes are faulty but operational. The method used is to obtain the time from all available servers, compute the best correct time interval, and see if any servers do not agree with the correct time. The details of the preferred algorithm shown in the flow chart of Fig. 17.

First, the time is obtained from all available servers including the time from the clock of the server node executing the method (Step 400). Time intervals obtained from servers of a different epoch (Step 402) than the executing server node (see below) are discarded and logged (Step 404).

The correct time interval corresponding to the resynchronization time T_r is computed using the algorithm presented in connection with the resynchronization of clerk nodes (Figs. 12-13), and the value of f, representing the number of faulty servers resulting in this interval, is recorded (Step 406). If the value of f is not zero, there are faulty servers (Step 408) and an event including the identification of all servers and their respective times is logged (Step 410). If the executing server's own time interval is not contained in the computed correct interval, then the executing server is faulty (Step 412). In this case, the executing server must refrain from responding to requests for the time until it has corrected itself (Step 414). A faulty time maintainer corrects itself by adjusting its clock as described above (for clerk nodes) in Figs. 8 and 9. A faulty timer provider attempts to correct itself by obtaining the correct time from the external reference.

Time maintainers preferably execute the procedure of Fig. 17 whenever they resynchronize. Time providers preferably execute the procedure at predetermined intervals.

vii) Leap Seconds in Server Nodes

The operation of a time maintainer node to accommodate leap seconds is identical to that of clerk nodes as described above with respect to Fig. 14. In some embodiments, time providers may include a local clock which is kept correct by periodically polling a device that receives the external signal (such as a WWV receiver). For the purpose of accomodating leap seconds, such a time provider node must make adjustments for leap seconds using the method used by time maintainers and clerks. More specifically, at the time of a potential leap second as measured by T + I, the inaccuracy must be increased by one second by a method similar to that of Fig. 14.

Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope and spirit of the invention being indicated by the following claims.

Claims

1. In a distributed processing system containing a plurality of interconnected nodes including a clerk node and a plurality of server nodes providing time to the system, wherein the clerk node includes a local clock having a predetermined resolution value, keeping a local time, and having an inaccuracy associated therewith representing an amount by which that local time deviates from a correct time value, a method for maintaining a correct time in the clerk node comprising the steps, performed by the clerk node, of:
requesting an updated time interval from at least one of the server nodes when the local inaccuracy of the local clock of the clerk node exceeds a predetermined maximum inaccuracy value;

ISDOCID: <EP___0350149A2_I_>

noting a request time according to the local clock of the clerk node when the clerk node requests said updated time interval;
receiving from the server nodes respective updated time interval representations and corresponding delay values;
noting, for each receipt of said updated time interval representations and corresponding delay values, a different response time according to the local clock of the clerk node;
calculating a correct time interval from said received updated time interval representations, said received delay values, said noted request time, said noted response times, and the resolution value of the local clock of the clerk node, wherein the correct time is contained within said calculated correct time interval; and
adjusting the local time kept by the local clock of the clerk node according to said calculated correct time interval.

2. The method of claim 1, further including the step of:
incrementing the local time of the local clock of the clerk node by periodically adding an incrementation value equal to the predetermined resolution value to the local time kept by the local clock, and
wherein said adjusting step further includes the step of:
changing said incrementation value according to said calculated correct time interval.

3. The method of claim 2, wherein said adjusting step further includes the steps of:
calculating a duration value to determine a length of time during which said changing step is performed; and
restoring said incrementation value to the predetermined resolution value when said incrementation value changing step has been performed for a length of time equal to said calculated duration value.

4. The method of claim 1, further including the step of:
calculating a resynchronization time at which the inaccuracy of the local clock of the clerk node exceeds said predetermined maximum inaccuracy value, and
wherein said requesting step is performed when the local clock of the clerk node has a time substantially equal to said calculated resynchronization time.

5. The method of claim 4, further including the step of requesting an updated time interval from the server nodes when the local clock has a time substantially equal to a predetermined maximum resynchronization time.

6. The method of claim 1, wherein each server node includes a local clock keeping a local time and having an inaccuracy value associated therewith, and wherein the server nodes include time maintainer node which updates its local clock by said requesting, request time noting, receiving, response time noting, calculating, and adjusting steps used by the clerk.node and a time provider node which updates its local clock by accessing an external time source, wherein said requesting step further includes the steps, performed by the clerk node, of:
requesting an updated time interval from the time maintainer node using a predetermined protocol; and
requesting an updated time interval from a time provider node using said predetermined protocol.

7. The method of claim 1, wherein each server node includes a local clock keeping a local time and having an inaccuracy value associated therewith, and wherein the server nodes include a time maintainer node which updates its local clock by and requesting, request time noting, receiving, response time noting, calculating, and adjusting steps used by the clerk node and a time provider node which updates its local clock by accessing an external time source, wherein said receiving step further includes the steps, performed by said clerk node, of:
receiving an updated time interval representation and a corresponding delay value from the time maintainer node using a predetermined protocol; and
receiving an updated time interval representation and corresponding delay values from the time provider node using said predetermined protocol.

8. In a distributed processing system containing a plurality of interconnected server nodes, one of which is called a time maintainer node, wherein the server nodes provide time to the system and wherein the time maintainer node includes a local clock having a predetermined resolution value, keeping a local time, and having an inaccuracy associated therewith representing an amount by which that local time deviates from a correct time value, a method for maintaining a correct time in the time maintainer node comprising the steps, performed by the time maintainer node, of:
requesting an updated time interval from at least one of the server nodes when the local inaccuracy of the local clock of the time maintainer node exceeds a predetermined maximum inaccuracy value;
noting a request time according to the local clock of the time maintainer node when the time maintainer node requests said updated time interval;
receiving from the server nodes respective updated time interval representations and corresponding delay

BNSDOCID: <EP__0350149A2_I_>

values;
noting, for each receipt of said updated time interval representations and corresponding delay values, a different response time according to the local clock of the time maintainer node;
calculating a correct time interval from said received updated time interval representations, said received delay values, said noted request time, said noted response times, and the resolution value of the local clock of the time maintainer node, wherein the correct time is contained within said calculated correct time interval; and
adjusting the local time kept by the local clock of the time maintainer node according to said calculated correct time interval.

9. The method of claim 8, further including the step of:
incrementing the local time of the local clock of the time maintainer node by periodically adding an incrementation value equal to the predetermined resolution value to the local time kept by the local clock, and
wherein said adjusting step further includes the step of:
changing said incrementation value according to said calculated correct time interval.

10. The method of claim 9, wherein said adjusting step further includes the steps of:
calculating a duration value to determine a length of time during which said changing step is performed; and
restoring said incrementation value to the predetermined resolution value when said incrementation value changing step has been performed for a length of time equal to the calculated duration value.

11. The method of claim 8, further including the step of:
calculating a resynchronization time at which the inaccuracy of the local clock of the time maintainer node exceeds said predetermined maximum inaccuracy value, and
wherein said requesting step is performed when the local clock of the time maintainer node has a time substantially equal to said calculated resynchronization time.

12. The method of claim 8, further including the step of requesting an updated time interval from the server nodes when the local clock has a time substantially equal to a predetermined maximum resynchronization time.

13. The method of claim 8, further including the step of detecting as faulty server nodes those server nodes from which updated time interval representations outside of said calculated correct time interval are received.

14. The method of claim 8, wherein each server node includes a local clock keeping a local time and having an inaccuracy value associated therewith, and wherein the server nodes include a second time maintainer node and a time provider node which updates its local clock by accessing an external time source, wherein said requesting step further includes the steps, performed by the time maintainer node, of:
requesting an updated time interval from the second time maintainer node using a predetermined protocol; and
requesting an updated time interval from the time provider node using said predetermined protocol.

15. The method of claim 8, wherein each server node includes a local clock keeping a local time and having an inaccuracy value associated therewith, and wherein the server nodes include a second time maintainer node and a time provider node which updates its local clock by accessing an external time source, wherein said receiving step further includes the steps, performed by the time maintainer node of, of:
receiving an updated time interval representation and a corresponding delay value from the second time maintainer node using a predetermined protocol; and
receiving an updated time interval representation and corresponding delay values from the time provider node using said predetermined protocol.

16. The method of claim 8, wherein each of the server nodes possesses one of a plurality of epoch numbers identifying a group of nodes,
wherein said time interval representation and delay value receiving step further includes the step of receiving from the server nodes the epoch numbers of the server nodes, and
further including the step of ignoring said updated time interval representations and corresponding delay values received from a server node when said received epoch number received from the server node differs from said epoch number of the time maintainer node.

17. In a distributed processing system containing interconnected nodes including a server node and one other node, wherein the nodes include a local clock having a predetermined resolution value, keeping a local time, and having an inaccuracy associated therewith representing an amount by which that local time deviates from a correct time value, a method for maintaining a correct time in the other node comprising the steps, performed by the server node, of:

VSDOCID: <EP___0350149A2_I_>

receiving a request for an updated time interval from the other node when the local inaccuracy of the local clock of the other node exceeds a predetermined maximum inaccuracy value;
noting a receipt time according to the local clock of the server node when the server node receives said request from the other node;
calculating a delay value from said noted receipt time;
calculating an updated time interval representation from the local time of the local clock of the server node and the inaccuracy of the local clock of the server node; and
sending said updated time interval representation and said corresponding delay value to the other node.

18. The method of claim 17, wherein the server node possesses one of a plurality of epoch numbers identifying a group of nodes, and
wherein said sending step further includes the step of sending the epoch number of the server node to the other node.

19. In a distributed processing system containing a plurality of interconnected nodes including a clerk node and a server node providing time to the system, wherein each of the nodes includes a local clock having a predetermined resolution value, keeping a local time, and having an inaccuracy associated therewith representing an amount by which that local time deviates from a correct time value, a method for maintaining a correct time in the clerk node comprising the steps of:
requesting, by the clerk node, an updated time interval from the server node when the local inaccuracy of the local clock of the clerk node exceeds a predetermined maximum inaccuracy value;
noting, by the clerk node, a request time according to the local clock of the clerk node when the clerk node requests said updated time interval;
receiving, by the clerk node, from the server node an updated time interval representation and corresponding delay value;
noting, by the clerk node, for said receipt of said updated time interval representation and corresponding delay value, a response time according to the local clock of the clerk node;
calculating, by the clerk node, a correct time interval from said received updated time interval representation, said received delay value, said noted request time, said noted response time, and the resolution value of the local clock of the clerk node, wherein the correct time is contained within said calculated correct time interval;

adjusting, by the clerk node, the local time kept by the local clock of the clerk node according to said calculated correct time interval;
receiving, by the server node, said request for an updated time interval from the clerk node;
noting, by the server node, a receipt time according to the local clock of the server node when the server node receives said request from the clerk node;
calculating, by the server node, a delay value from said noted receipt time;
calculating, by the server node, an updated time interval representation from the local time of the local clock of the server node and the inaccuracy of the local clock of the server node; and
sending, by the server node, said updated time interval representation and said corresponding delay value to the clerk node.

20. The method of claim 19, wherein the server node possesses one of a plurality of epoch numbers identifying a group of nodes, and
wherein said sending step further includes the step, performed by the server node, of sending the epoch number of the server node to the clerk node.

21. The method of claim 19, further including the step, performed by the clerk node, of:
incrementing the local time of the local clock of the clerk node by periodically adding an incrementation value equal to the predetermined resolution value to the local time kept by the local clock of the clerk node, and
wherein said adjusting step further includes the step, performed by the clerk node, of changing said incrementation value according to said calculated correct time interval.

22. The method of claim 21, wherein said adjusting step further includes the steps, performed by the clerk node, of:
calculating a duration value to determine a length of time during which said changing step is performed; and
restoring said incrementation value to the predetermined resolution value when said changing step has been performed for a length of time equal to said calculated duration value.

23. The method of claim 19, further including the step, performed by the clerk node, of calculating a resynchronization time at which the inaccuracy of the local clock of the clerk node exceeds said

BNSDOCID: <EP__0350149A2_I_>

predetermined maximum inaccuracy value, and
wherein said requesting step is performed by the clerk node when the local clock of the clerk node has a time substantially equal to said calculated resynchronization time.

24. The method of claim 23, further including the step, performed by the clerk node, of requesting an updated time interval from the server node when the local clock of the clerk node has a time substantially equal to a predetermined maximum resynchronization time.

25. The method of claim 19, wherein the distributed processing further contains a plurality of interconnected server nodes including a time maintainer node which updates its local clock by said requesting, request time noting, receiving, response time noting, calculating, and adjusting steps used by the clerk node and a time provider node which updates its local clock by accessing an external time source, wherein said requesting step performed by the clerk node further includes the steps of:
requesting an updated time interval from the time maintainer node using a predetermined protocol; and
requesting an updated time interval from the time provider node using said predetermined protocol.

26. The method of claim 19, wherein the distributed data processing system further contains a plurality of interconnected server nodes including a time maintainer node which updates its local clock by said requesting, request time noting, receiving, response time noting, calculating, and adjusting steps used by the clerk node and a time provider node which updates its local clock by accessing an external time source, wherein said receiving step performed by the clerk node further includes the steps of:
receiving an updated time interval representation and a corresponding delay value from the time maintainer node using a predetermined protocol; and
receiving an updated time interval representation and a corresponding delay value from the time provider node using said predetermined protocol.

27. The method of claim 19, wherein the server node is a time maintainer node, wherein the distributed processing system further contains a second server node providing time to the system, and further comprising the steps, performed by the time maintainer node, of:
requesting an updated time interval from the other server node when the local inaccuracy of the local clock of the time maintainer node exceeds a predetermined maximum inaccuracy value;
noting a request time according to the local clock of the time maintainer node when the time maintainer node requests said updated time interval;
receiving from the other server node an updated time interval representation and a corresponding delay value;
noting, for said receipt of said updated time interval representation and corresponding delay value, a response time according to the local clock of the time maintainer node;
calculating a correct time interval from said received updated time interval representation, said received delay value, said noted request time, said noted response time, and the resolution value of the local clock of the time maintainer node, wherein the correct time is contained within said calculated correct time interval; and
adjusting the local time kept by the local clock of the time maintainer node according to said calculated correct time interval.

28. The method of claim 27, further including the step, performed by the time maintainer node, of:
incrementing the local time of the local clock of the time maintainer node by periodically adding an incrementation value equal to the predetermined resolution value to the local time kept by the local clock, and
wherein said adjusting step performed by the time maintainer node further includes the step of changing said incrementation value according to said calculated correct time interval.

29. The method of claim 28, wherein said adjusting step performed by the time maintainer node further includes the steps of:
calculating a duration value to determine a length of time during which said changing step is performed; and
restoring said incrementation value to the predetermined resolution value when said incrementation value changing step has been performed for a length of time equal to the calculated duration value.

30. The method of claim 27, further including the step, performed by the time maintainer node, of:
calculating a resynchronization time at which the inaccuracy of the local clock of the time maintainer node exceeds said predetermined maximum inaccuracy value, and
wherein said requesting step is performed by the time maintainer node when the local clock of the time maintainer node has a time substantially equal to said calculated resynchronization time.

31. The method of claim 27, further including the step, performed by the time maintainer node, of requesting an updated time interval from the other server node when the local clock has a time substantially

ISDOCID: <EP___0350149A2_I_>

equal to a predetermined maximum resynchronization time.

32. The method of claim 27, further including the step, performed by the time maintainer node, of detecting as faulty server nodes those server nodes from which updated time interval representations outside of said calculated correct time interval are received.

33. The method of claim 27, wherein the other server node is a time provider node which updates its local clock by accessing an external time source, wherein the distributed processing system further contains a second time maintainer node providing time to the system, and wherein said requesting step performed by the time maintainer node further includes the steps of:
requesting an updated time interval from the second time maintainer node using a predetermined protocol; and
requesting an updated time interval from the time provider node using said predetermined protocol.

34. The method of claim 27, wherein the other server node is a time provider node which updates its local clock by accessing an external time source, wherein the distributed processing system further contains a second time maintainer node providing time to the system, and wherein said requesting step performed by the time maintainer node further includes the steps of:
receiving an updated time interval representation and a corresponding delay value from the second time maintainer node using a predetermined protocol; and
receiving an updated time interval representation and corresponding delay values from the time provider node using said predetermined protocol.

35. The method of claim 27, wherein each of the server nodes possesses one of a plurality of epoch numbers identifying a group of nodes,
wherein said time interval representation and delay value receiving step performed by the time maintainer node further includes the step of receiving from the other server nodes the epoch numbers of the server nodes, and
further including the step of ignoring said updated time interval representations and corresponding delay values received from a server node when said received epoch number received from said server node differs from said epoch number of the time maintainer node.

36. The method of claim 19, wherein the server node is a time provider node which updates its local clock by accessing an external time source, wherein the distributed processing system further contains an other server node providing time to the system, and further including the steps, performed by the time provider node, of:

requesting an updated time interval from the other server node periodically;
noting a request time according to the local clock of the time provider node when the time provider node requests said updated time interval;
receiving from the other server node an updated time interval representation and corresponding delay value;
noting, for said receipt of said updated time interval representation and corresponding delay value, a response time according to the local clock of the time provider node;
calculating a correct time interval from said received updated time interval representation, said received delay value, said noted request time, said noted response time, and the resolution value of the local clock of the time provider node, wherein the correct time is contained within said calculated correct time interval; and
detecting as faulty server nodes those server nodes from which updated time interval representations outside of said calculated correct time interval are received.

37. A clerk node in a distributed processing system containing a plurality of interconnected server nodes, the clerk node providing a correct time to an application program in the system and comprising:
a memory having a location accessible by the application program;
a local clock keeping a local time having an associated inaccuracy;
incrementing means connected to said local clock for causing said local time to increase periodically;
first processing means connected to at least one server node and responsive to a first series of instructions stored in said memory, for ensuring that said local time of said local clock is within a predetermined inaccuracy of the correct time, said processing means including:
means for requesting an updated time interval from at least one of the server nodes when the inaccuracy of said local clock exceeds a predetermined maximum inaccuracy value,
means for noting a request time according to said local clock when the clerk node requests said updated time interval,
means for receiving from the server nodes respective updated time interval representations and cor-

responding delay values,
means for noting, for each receipt of said updated time interval representations and corresponding delay values, a different response time according to said local clock,
means for calculating a correct time interval from said received updated time interval representations, said received delay values, said noted request time, said noted response times, and said resolution value of said local clock, wherein the correct time is contained within said calculated correct time interval, and
means for adjusting said local time kept by said local clock according to said calculated correct time interval; and
second processing means, connected to said memory and to said local clock and responsive to a second series of instructions stored in said memory, for storing said local time kept by said local clock into the memory location accessible by the application program.

38. A time maintainer node in a distributed processing system containing a plurality of interconnected server nodes providing time information to the system, the time maintainer node maintaining a correct time and comprising:
a memory;
a local clock keeping a local time having an associated inaccuracy;
incrementing circuit means connected to said local clock for causing said local time of said local clock to increase periodically; and
processing means connected to at least one server node and responsive to a series of instructions stored in said memory, for ensuring that said local time of said local clock is within a predetermined inaccuracy of the correct time, said processing means including:
means for requesting an updated time interval from at least one of the server nodes when the inaccuracy of said local clock exceeds said predetermined maximum inaccuracy value,
means for noting a request time according to said local clock when the time maintainer node requests said updated time interval,
means for receiving from the server nodes respective updated time interval representations and corresponding delay values,
means for noting, for each receipt of said updated time interval representations and corresponding delay values, a different response time according to said local clock,
means for calculating a correct time interval from said received updated time inrerval representations, said received delay values, said noted request time, said noted response times, and said resolution value of said local clock, wherein the correct time is contained within said calculated correct time interval, and
means for adjusting the local time kept by said local clock according to said calculated correct time interval.

39. The time maintainer node of claim 38, wherein the distributed processing system further includes a requesting node that requests an updated time interval,
and wherein the time maintainer node further includes:
second processing means, connected to the requesting node and to said local clock, responsive to a second series of instructions stored in said memory, for responding to a request from the requesting node, said second processing means including:
means for receiving the request for an updated time interval from the requesting node,
means for noting a receipt time according to said local clock when the time maintainer node receives the request from the requesting node,
means for calculating a delay value from said noted receipt time,
means for calculating an updated time interval representation from the local time of said local clock and the inaccuracy of said local clock, and
means for sending said updated time interval representation and said corresponding delay value to the requesting node.

40. The time maintainer node of claim 39, wherein said time maintainer node possesses one of a plurality of epoch numbers identifying a group of nodes, and
wherein said second processing means further includes means for sending the epoch number of the time maintainer node to the requesting node.

41. A time provider node in a distributed processing system containing a plurality of interconnected server nodes, the time provider node maintaining a correct time and comprising:
a memory;
a local clock keeping a local time having an associated inaccuracy;
increment circuit means connected to said local clock for causing said local time to increase periodically;
external time source interface means for receiving the correct time from an external time source; and
processing means connected to said external time source interface means and executing instructions stored

SDOCID: <EP___0350149A2_I_>

in the memory for adjusting the local time kept by said local clock according to said received correct time.

42. The time provider node of claim 41, wherein the distributed processing system further includes a requesting node that requests an updated time interval, wherein the time provider node further includes:

second processing means connected to said local clock and to the requesting node, responsive to a second series of instructions stored in said memory, for responding to the request for an updated time interval, said second processing means including:

means for receiving the request for an updated time interval from the requesting node,

means for noting a receipt time according to said local clock when the time provider node receives said request from the requesting node,

means for calculating a delay value from said noted receipt time,

means for calculating an updated time interval representation from the local time of said local clock and the inaccuracy of the local clock, and

means for sending said updated time interval representation and said corresponding delay value to the requesting node.

43. The time provider node of claim 42, wherein the time provider node possesses one of a plurality of epoch numbers identifying a group of nodes, and

wherein said second processing means further includes means for sending the epoch number of the time provider node to the requesting node.

RNSDOCID: <EP 0350149A2 I..>

FIG. 1

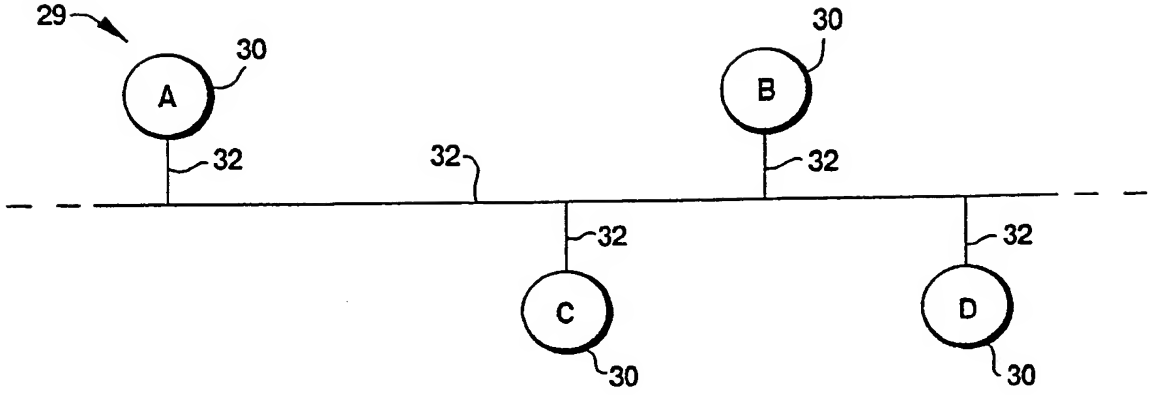
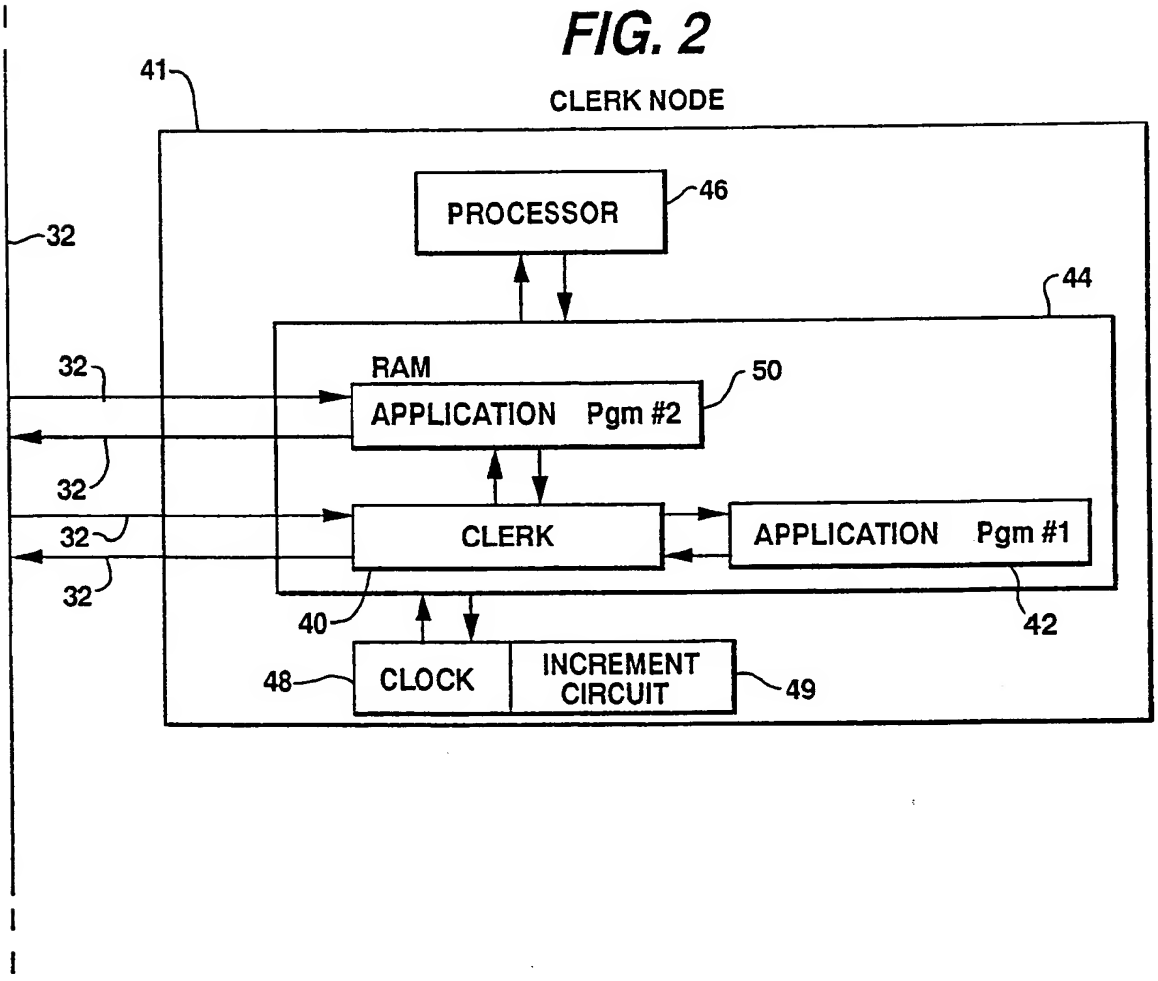


FIG. 2



SDOCID: <EP___0350149A2_I_>

FIG. 3

NOTATION ASSOCIATED WITH A TIME INTERVAL

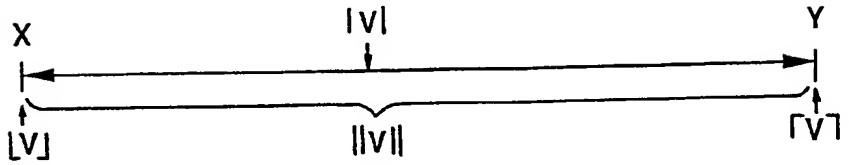
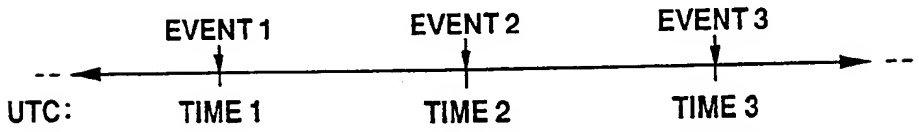
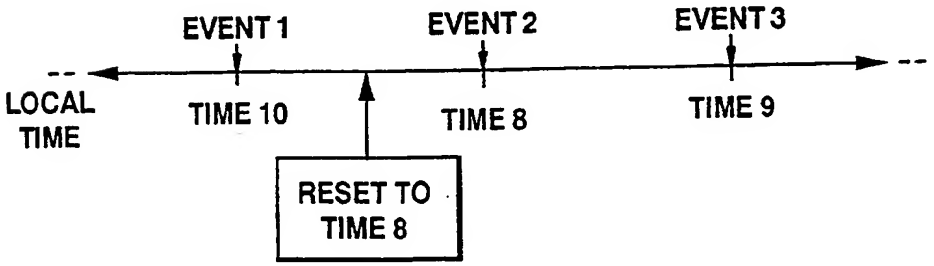


FIG. 4a



ORDERING BY UTC: <EVENT 1, EVENT 2, EVENT 3>

FIG. 4b



ORDERING BY LOCAL TIME: <EVENT 2, EVENT 3, EVENT 1>

FIG. 5 a

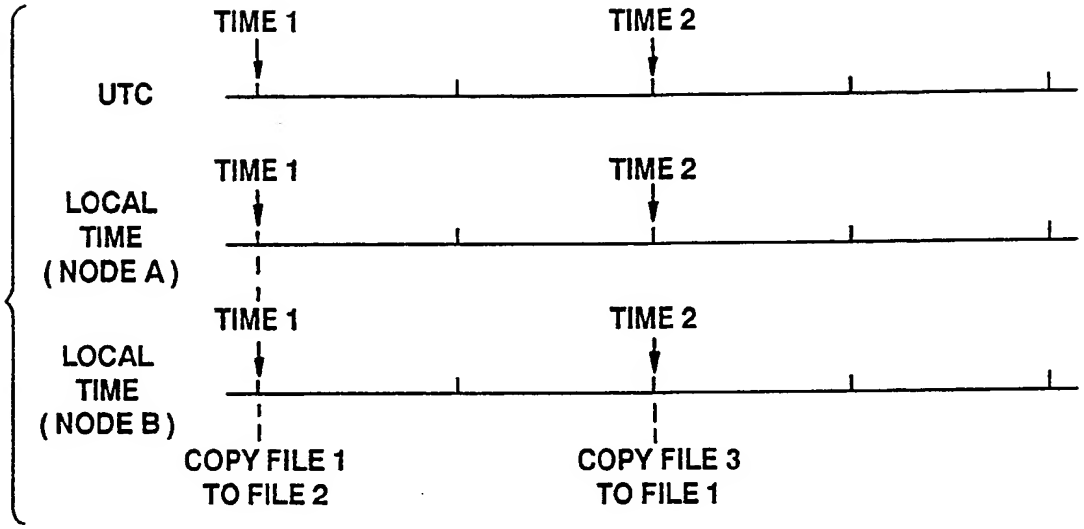
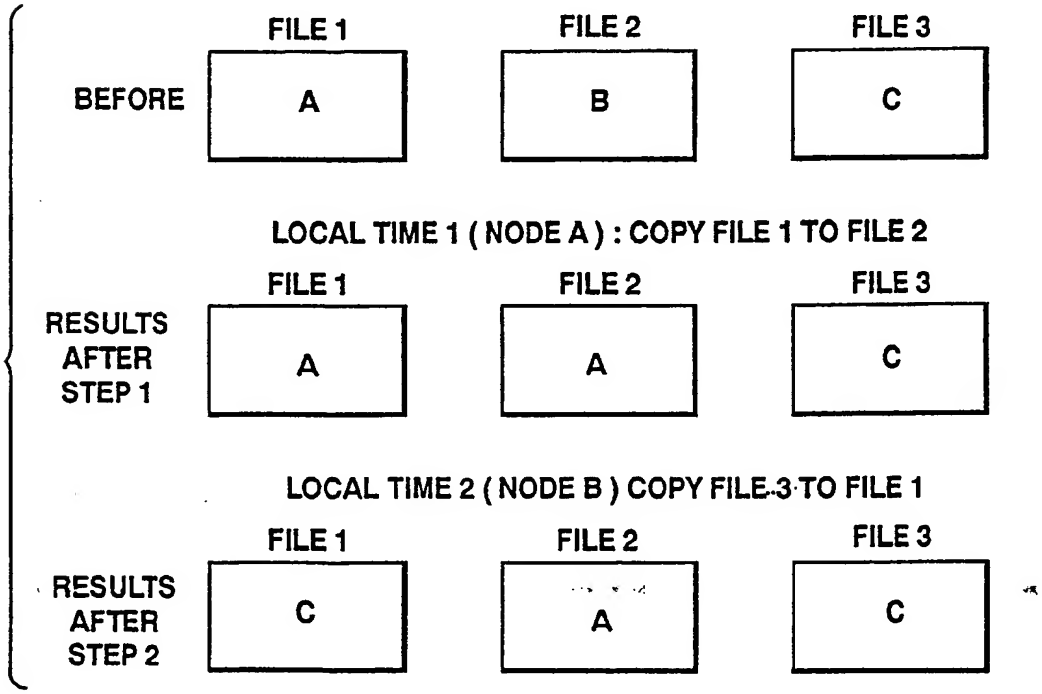


FIG. 5b



ISDOCID: <EP___0350149A2_I_>

FIG. 5 c

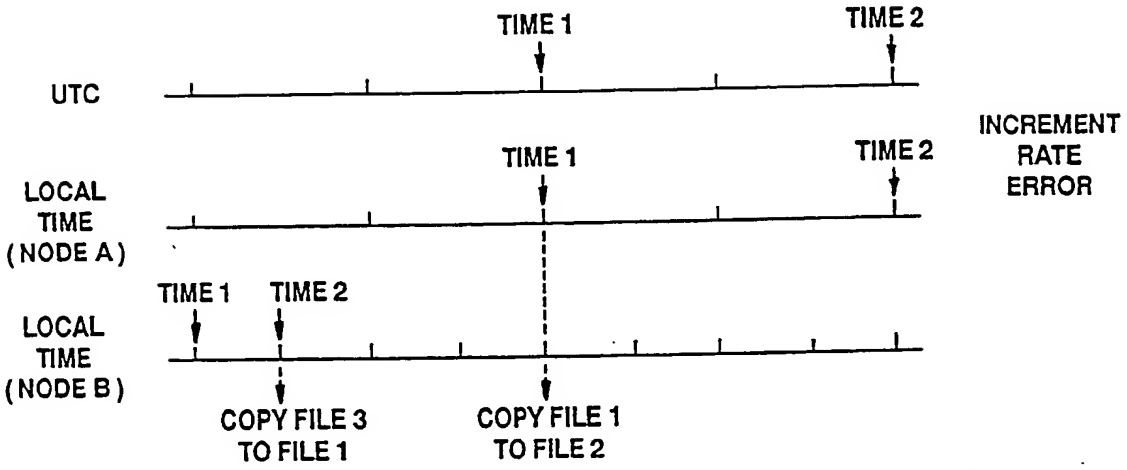


FIG. 5 d

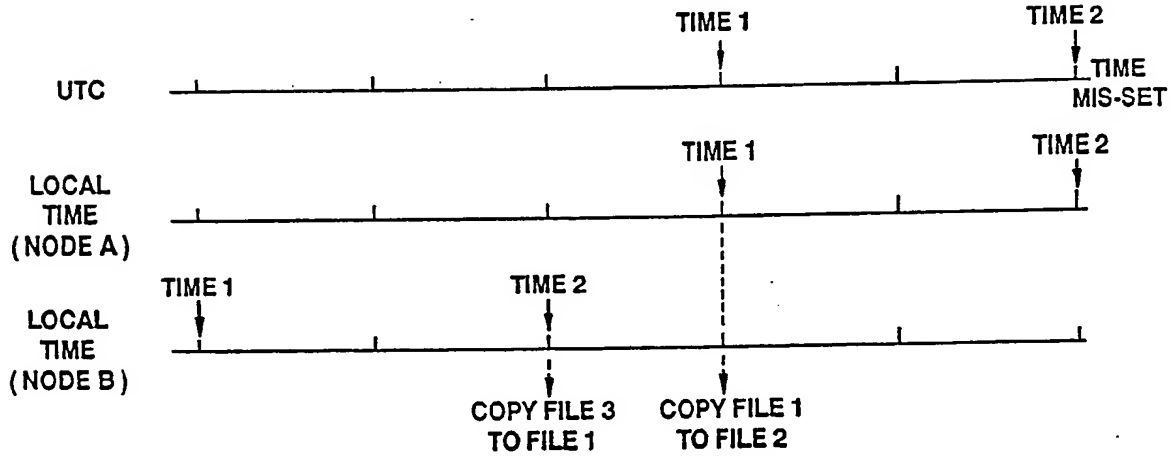
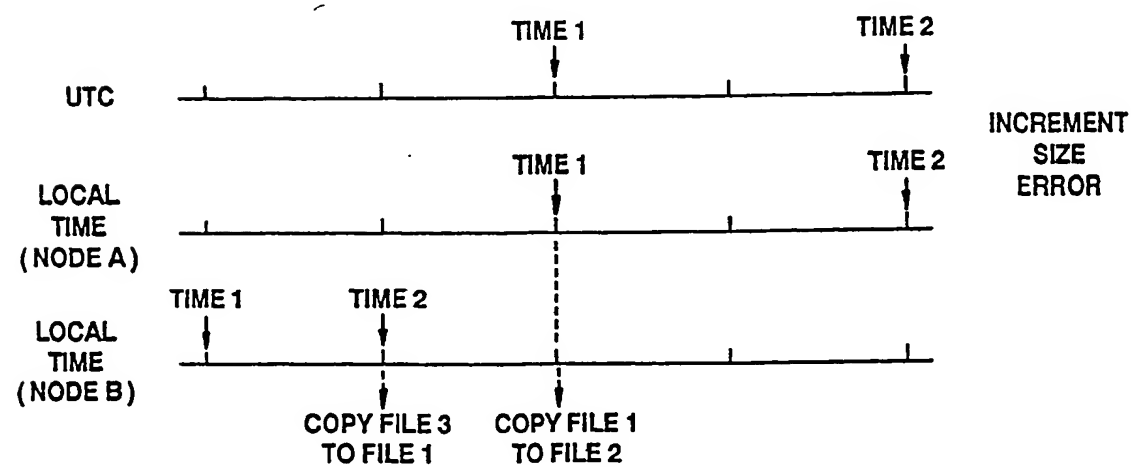
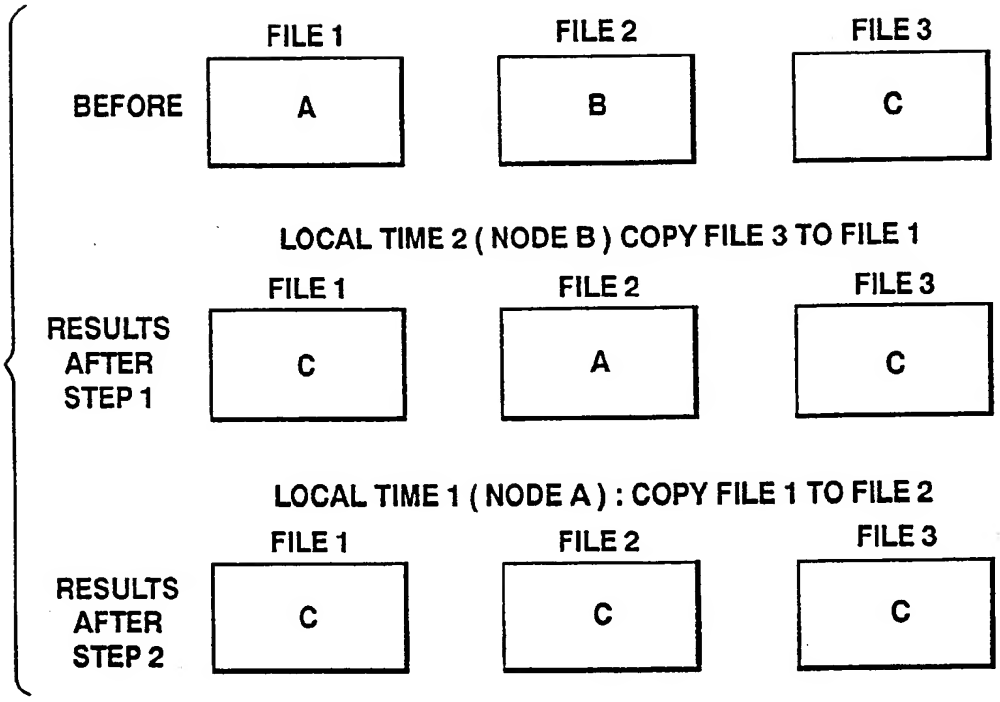


FIG. 5 e



BNSDOCID: <EP___0350149A2_I_>

FIG. 5f



NSDOCID: <EP___0350149A2_I_>

FIG. 6

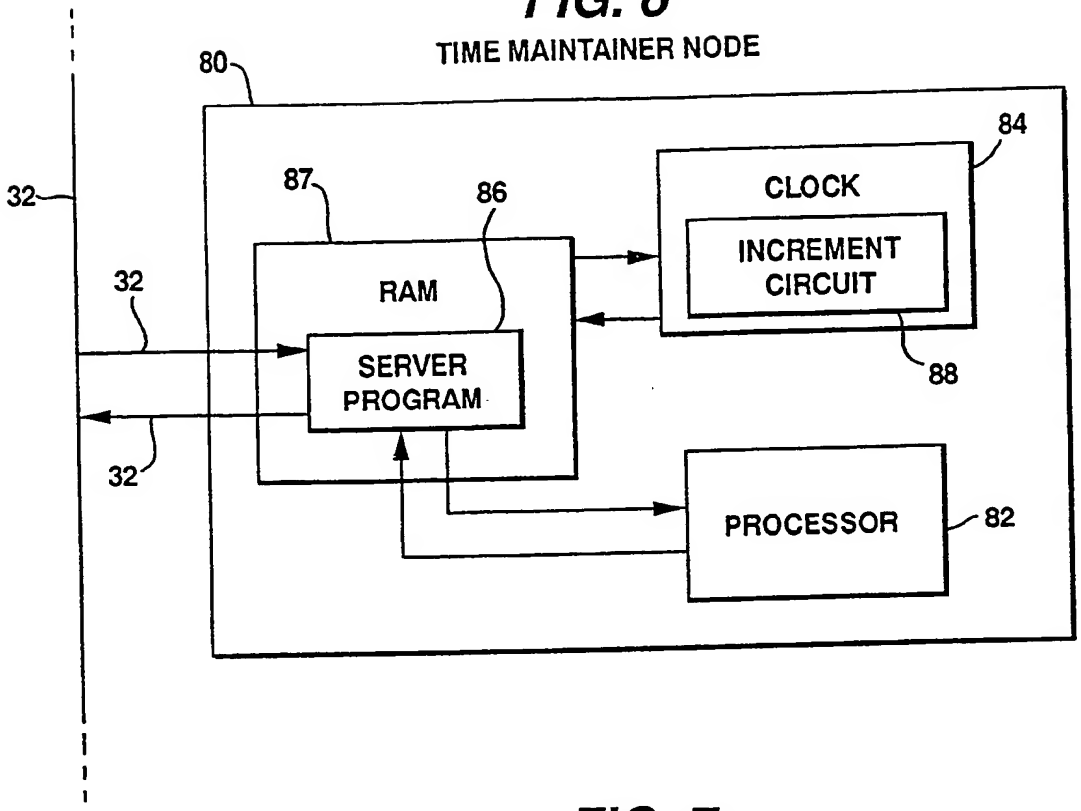

TIME MAINTAINER NODE
80
84
87
86
32
CLOCK
INCREMENT CIRCUIT
RAM
32
SERVER PROGRAM
88
32
PROCESSOR
82

FIG. 7

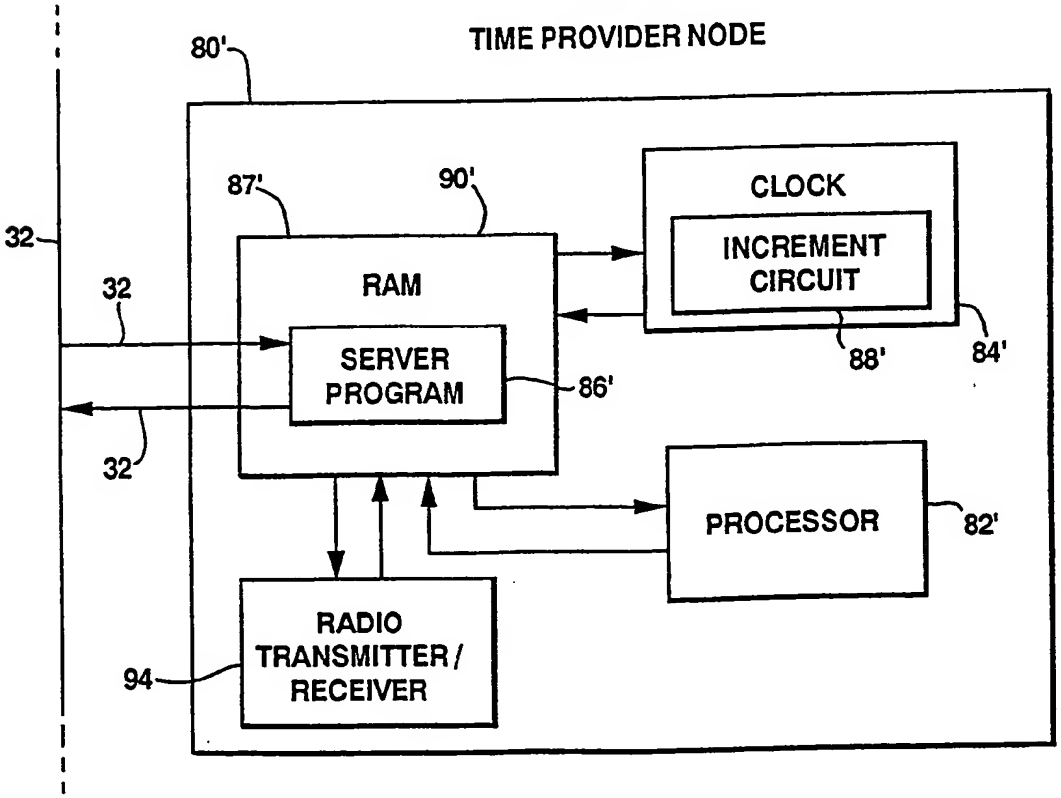
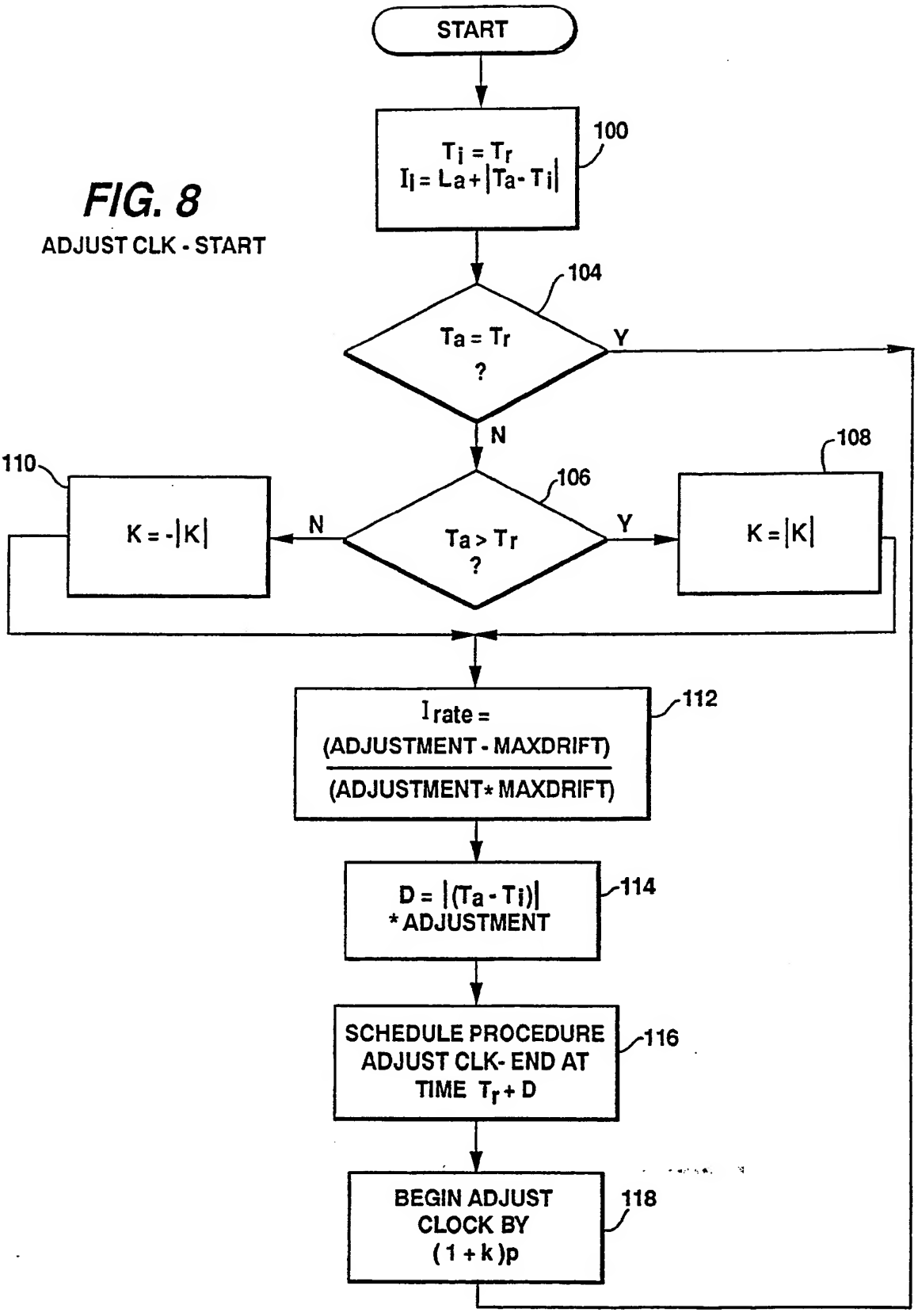

TIME PROVIDER NODE
80'
87'
90'
CLOCK
32
INCREMENT CIRCUIT
RAM
32
88'
84'
SERVER PROGRAM
86'
32
PROCESSOR
82'
RADIO TRANSMITTER / RECEIVER
94

FIG. 8

ADJUST CLK - START



SDOCID: <EP___0350149A2_I_>

FIG 9

ADJUST CLK - END

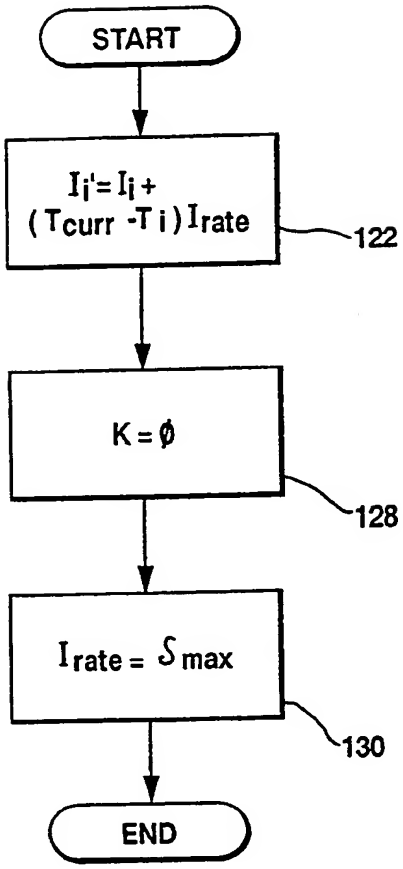
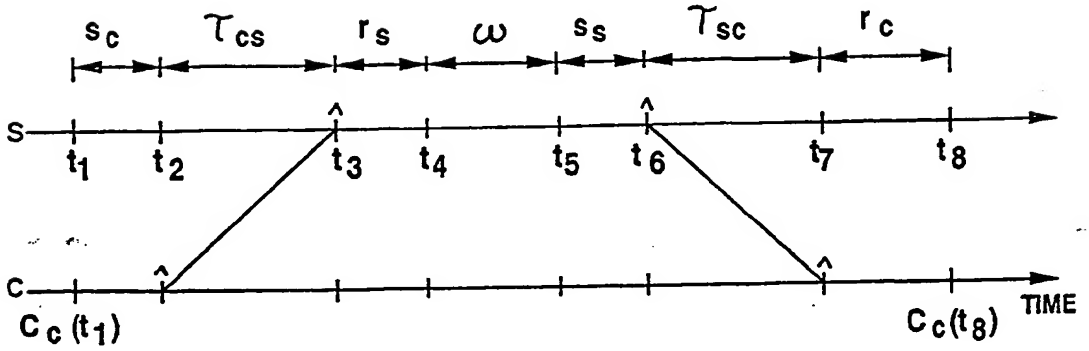


FIG. 10



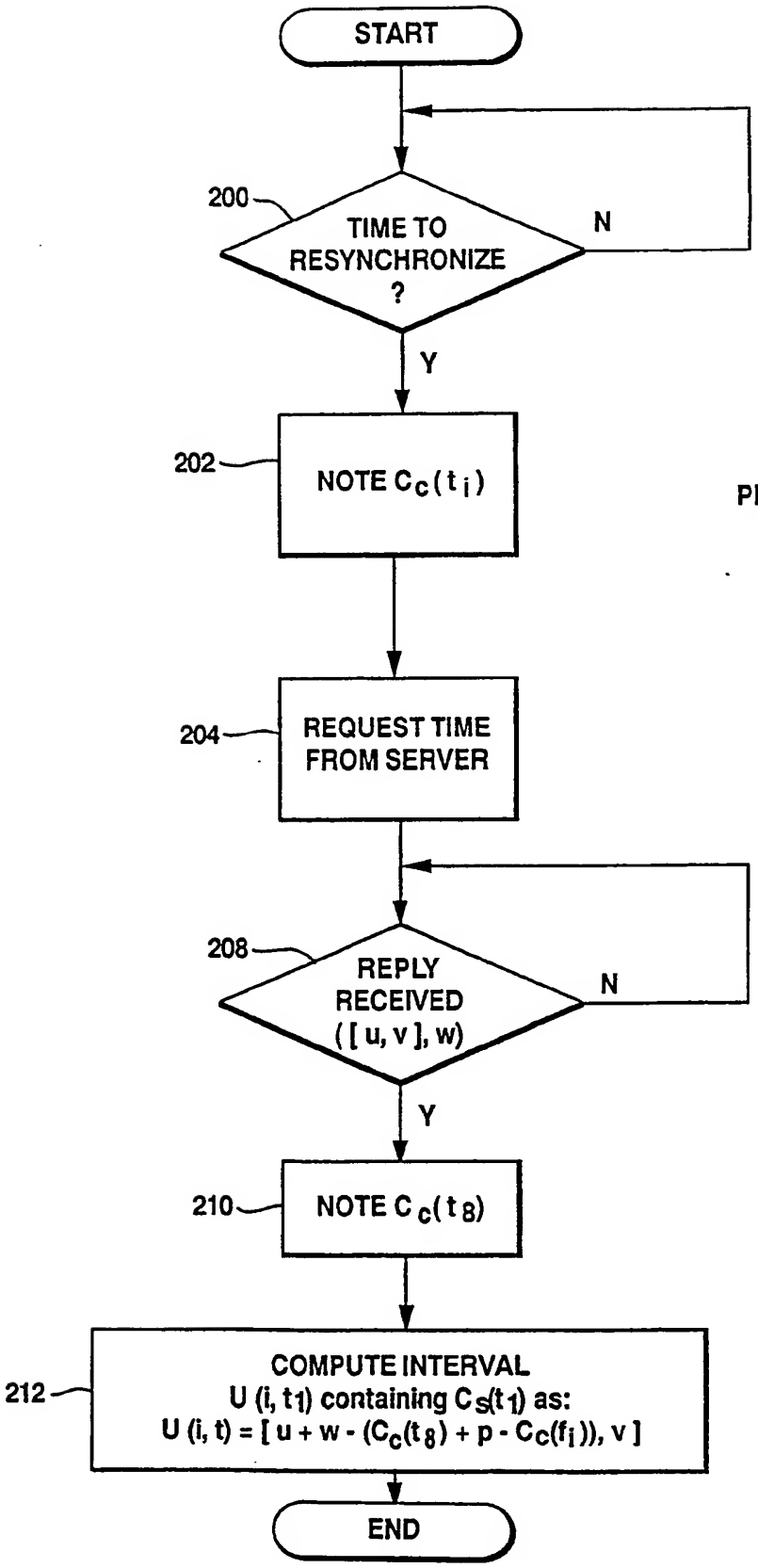


FIG. 11

PERFORMED BY CLERK

SDOCID: <EP___0350149A2_I_>

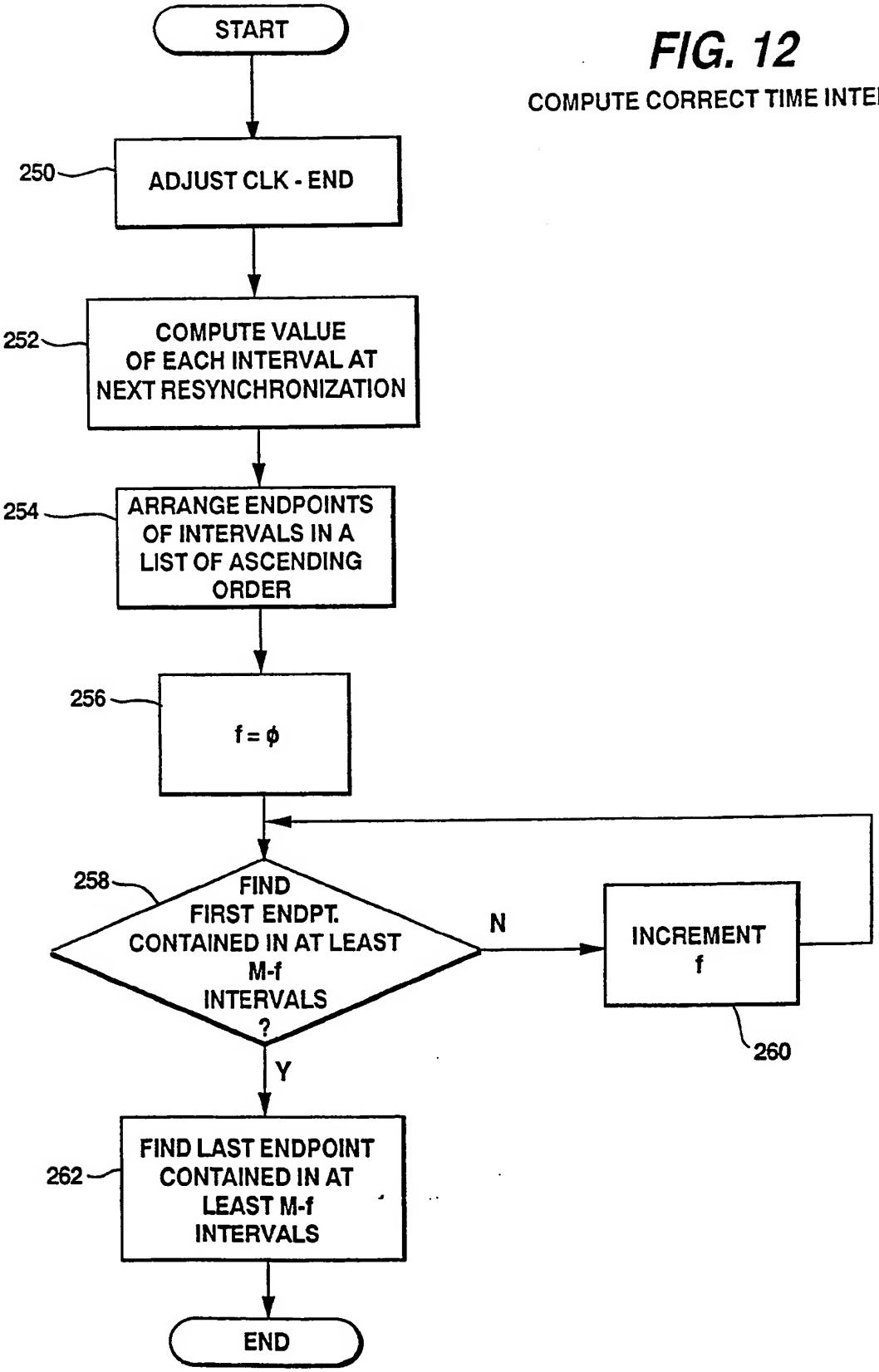


FIG. 12

COMPUTE CORRECT TIME INTERVAL

BNSDOCID: <EP___0350149A2_I_>

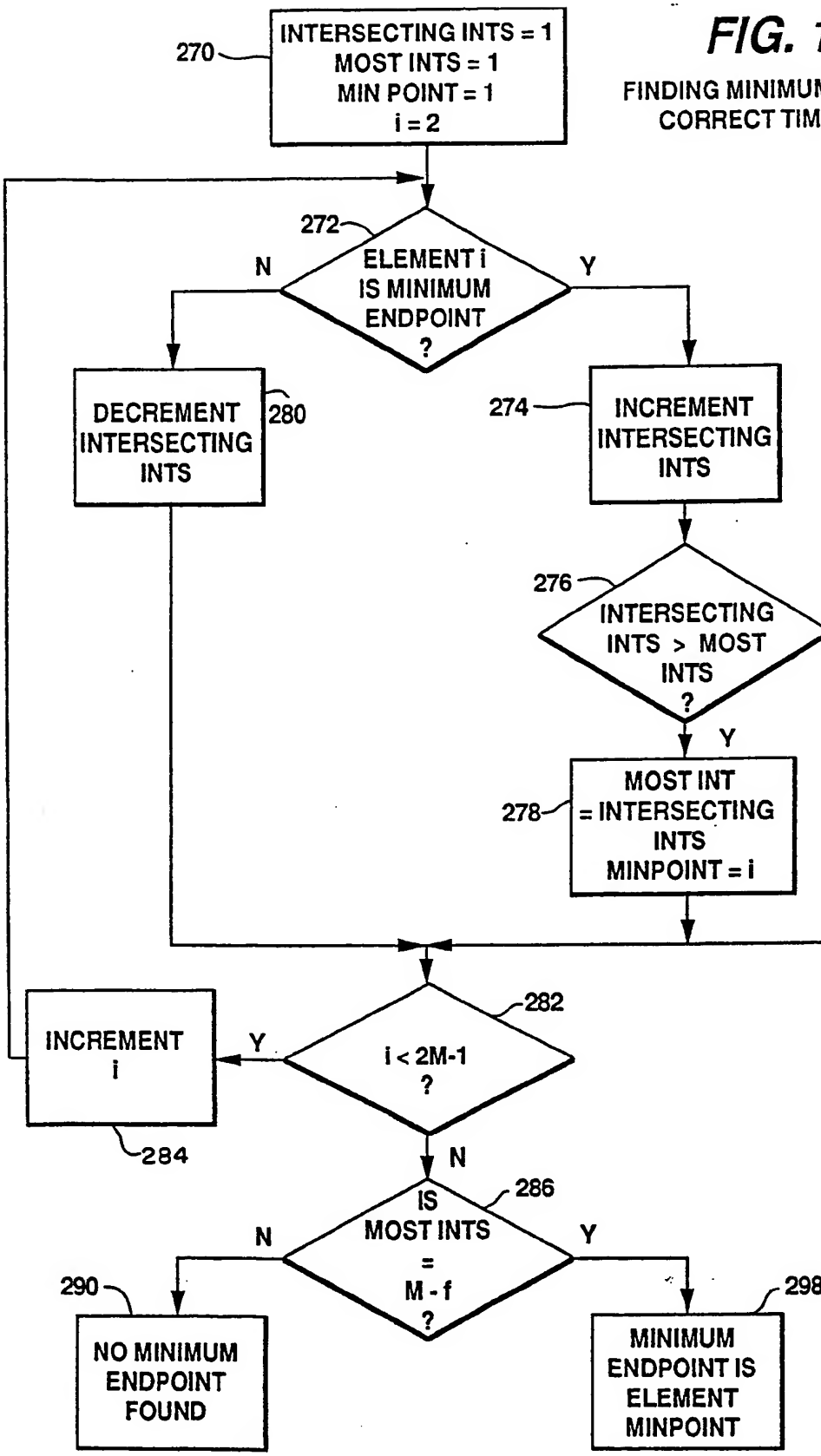


FIG. 13

FINDING MINIMUM ENDPOINT OF CORRECT TIME INTERVAL

ISDOCID: <EP___0350149A2_I_>

FIG. 15

PERFORMED BY SERVER

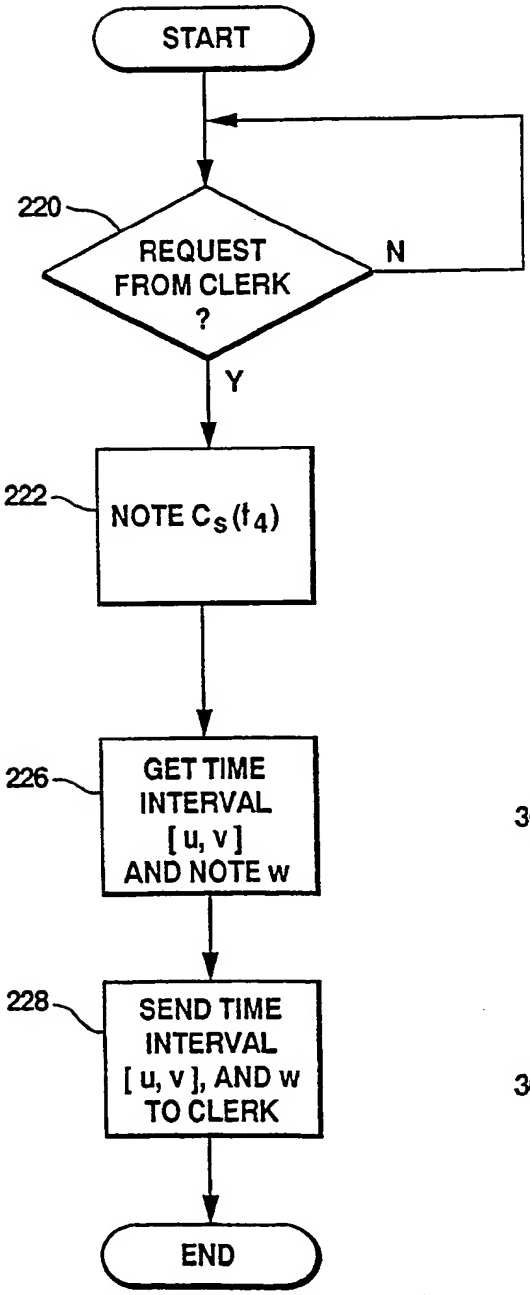
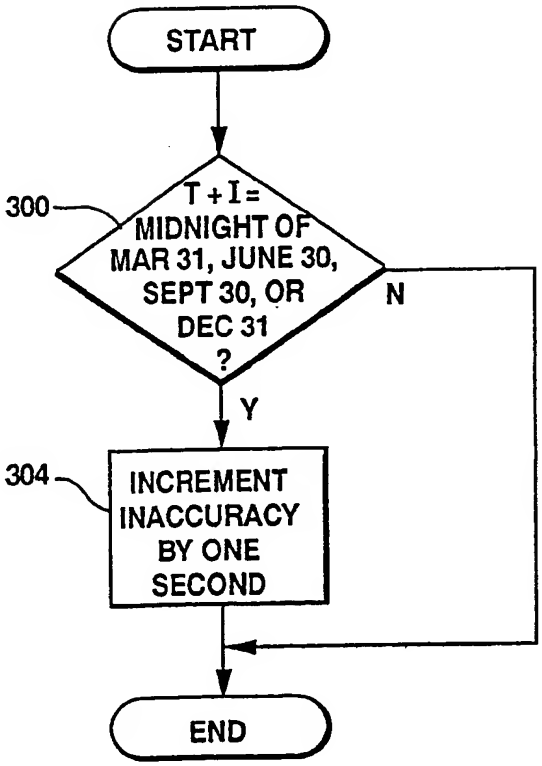


FIG. 14

ADJUSTMENT OF A LOCAL CLOCK FOR LEAP SECONDS



BNSDOCID: <EP___0350149A2_I_>

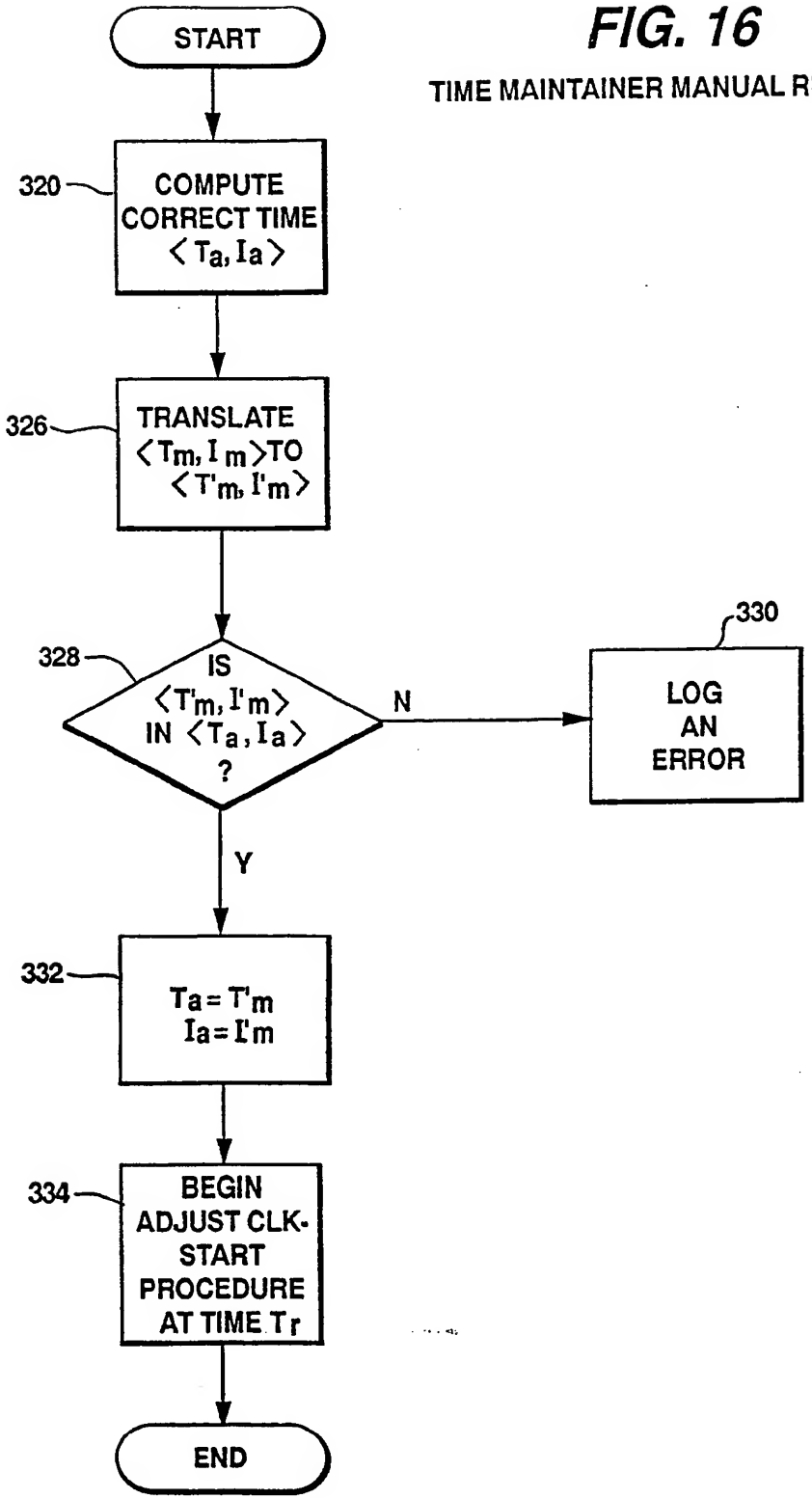


FIG. 16

TIME MAINTAINER MANUAL RESET

NSDOCID: <EP___0350149A2_I_>

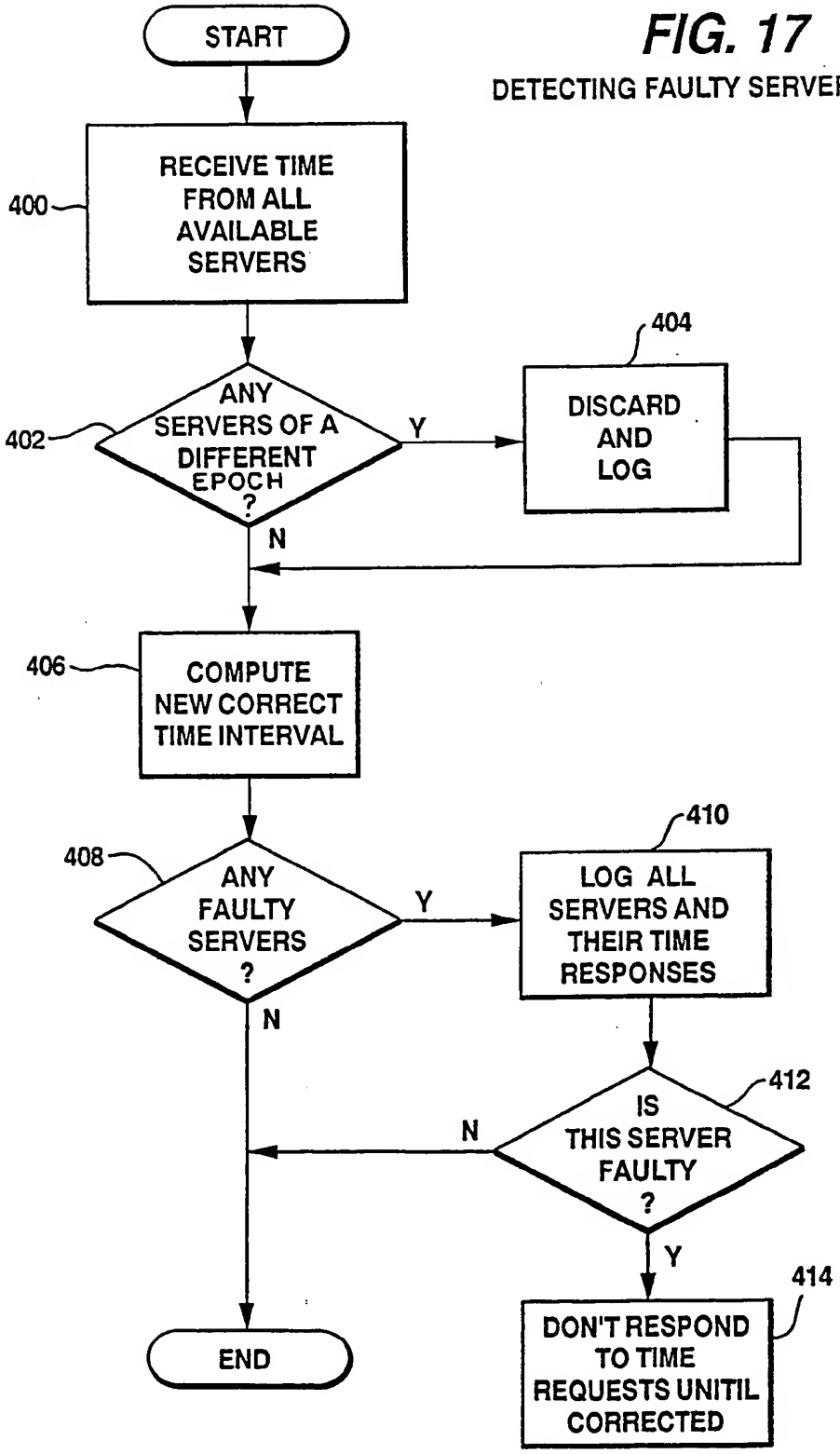


FIG. 17

DETECTING FAULTY SERVER NODES

BNSDOCID: <EP___0350149A2_I_>

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 350 149 A3**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **89304046.9**

(51) Int. Cl.5: **H04J 3/06**, G06F 1/14

(22) Date of filing: **24.04.89**

(30) Priority: **30.06.88 US 213746**

(43) Date of publication of application:
10.01.90 Bulletin 90/02

(84) Designated Contracting States:
DE FR GB IT NL

(88) Date of deferred publication of the search report:
28.10.92 Bulletin 92/44

(71) Applicant: **DIGITAL EQUIPMENT CORPORATION**
146 Main Street
Maynard, MA 01754(US)

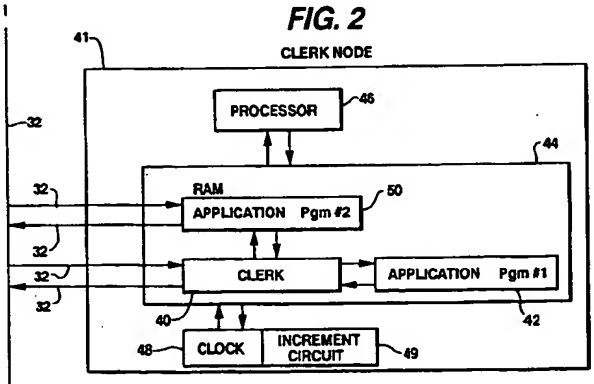
(72) Inventor: **Fine, Michael**
1 Ballord Place
Cambridge, MA 07139(US)

(74) Representative: **Goodman, Christopher et al**
Eric Potter & Clarkson St. Mary's Court St. Mary's Gate
Nottingham NG1 1LE(GB)

(54) Method of maintaining a correct time in a distributed processing system.

(57) A method for maintaining a correct time in a distributed processing system involves clerk nodes maintaining their local clocks by requesting time intervals from server nodes, and server nodes maintaining their local clocks either by requesting time intervals from other server nodes or by receiving time information from an outside source. Server nodes also provide time intervals to requesting clerk nodes and requesting server nodes with such a method. System time is always increasing and monotonic and faulty servers are detected periodically.

FIG. 2



EP 0 350 149 A3

BNSDOCID: <EP___0350149A3_I_>

European Patent Office

EUROPEAN SEARCH REPORT

Application Number EP 89 30 4046

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
X	IEEE TRANSACTIONS ON COMPUTERS vol. C-36, no. 8, August 1987, pages 933 - 940; H. KOPETZ ET AL: 'Clock Synchronization in Distributed Systems' * abstract *	1,2,6-9, 14,15, 17, 19-21, 25-28, 33,34 36-39, 41,42	H04J3/06 G06F1/14
A	* page 934, left column, line 30 - line 49 * * page 935, left column, line 20 - right column, line 5 * * page 935, right column, line 41 - page 936, right column, line 2 * * page 938, right column, line 27 - page 939, right column, line 3; figures 1-3 *	3-5, 10-13, 16, 22-24, 29-32, 35,40,43	
A	CONFERENCE RECORD, 1976 INTERNATIONAL CONFERENCE ON COMMUNICATIONS, 14-16 JUNE 1976, VOL. 2 PAPER 25 PAGES 1-6, PHILADELPHIA US H.A. STOVER: 'Timing and Synchronization of Switched Digital Communications Networks' * page 3, left column, line 36 - page 4, right column, line 48 *	1,2,6-9, 14,15, 17, 19-21, 25-28, 33,34 36-39, 41,42	TECHNICAL FIELDS SEARCHED (Int. Cl.5) H04J G06F
A	US-A-4 746 920 (NELLEN ET AL) * abstract * * column 7, line 52 - column 8, line 49; figure 5 *	16,18, 20,35, 40,43	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	21 JULY 1992	O'REILLY

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P0401)

INSDOCID: <EP___0350149A3_I_>